Implementación de un Sistema Automatizado de Reentrenamiento mediante la Detección de Data Drift en un Ciclo MLOps para Entornos Big Data

Leyton Jean Piere Castro Clavijo

Proyecto de grado entregado para obtener el título de
Magister en Ingeniería de Software

Dirigida por
MSc. Felipe Buitrago Carmona

Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias
Maestría en Ingeniería de Software
Santiago de Cali
30 de noviembre de 2025

Santiago de Cali, 30 de noviembre de 2025.

Señores
Pontificia Universidad Javeriana Cali.
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software.
Cali.

Cordial Saludo.

Por medio de la presente hago constar que en mi calidad de director de trabajo de grado he revisado el proyecto titulado "Implementación de un Sistema Automatizado de Reentrenamiento mediante la Detección de Data Drift en un Ciclo MLOps para Entornos Big Data" realizado por el estudiante de Magister en Ingeniería de Software Leyton Jean Piere Castro Clavijo (cod: 1234640402), el cual se encuentra terminado y considero que cumple con los requisitos para ser sustentado.
Atentamente,

Felipe Buitrago Carmona

MSc. Felipe Buitrago Carmona

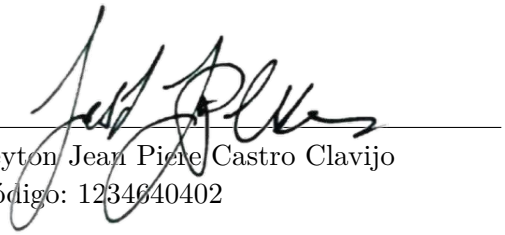
Santiago de Cali, 30 de noviembre de 2025.

Señores
Pontificia Universidad Javeriana Cali
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software
Cali.

Cordial Saludo.

Me permito presentar a su consideración el proyecto de grado titulado "Implementación de un Sistema Automatizado de Reentrenamiento mediante la Detección de Data Drift en un Ciclo MLOps para Entornos Big Data" con el fin de cumplir con los requisitos exigidos por la Universidad y para que sea sometido a revisión del jurado y cumpla su aprobación, para conseguir posteriormente el título de Magister en Ingeniería de Software.

Atentamente,



Leyton Jean Pierre Castro Clavijo
Código: 1234640402

Ficha Resumen
Trabajo de Grado Maestría en Ingeniería de Software

TíTULO: Implementación de un Sistema Automatizado de Reentrenamiento mediante la Detección de Data Drift en un Ciclo MLOps para Entornos Big Data

1. Énfasis: Ingeniería de Software
2. Área de trabajo: Ciencias de la Computación y la Ingeniería de Sistemas, con un enfoque específico en Big Data, Machine Learning y MLOps
3. Tipo de proyecto: Aplicado
4. Estudiante: Leyton Jean Piere Castro Clavijo
5. Correo electrónico: leytoncastro@javerianacali.edu.co
6. Dirección y teléfono: Cr. 9A #43-38 Ibagué Tolima, 318 785 7029
7. Director: Msc. Felipe Buitrago Carmona
8. Vinculación del director: Externo
9. Correo electrónico del director: felipe.buitrago@ucaldas.edu.co
10. Co-Director: NA
11. Grupo o empresa que lo avala (Si aplica):
12. Otros grupos o empresas: Ninguno
13. Palabras clave: Data Drift, MLOps, Big Data, Automatización, Machine Learning
14. ODS que aplica al proyecto (Agenda 2030): ODS 8, ODS 9, ODS 12.
15. Fecha de inicio: 23 de Junio de 2025
16. Resumen: El presente proyecto propone un sistema automatizado de reentrenamiento de modelos de *machine learning* que responde a la detección temprana del *data drift* en entornos Big Data. En la actualidad, las organizaciones enfrentan pérdidas promedio de 12,9 millones de USD anuales por mala calidad de datos y 4,88 millones por brechas de información (Gartner Inc., 2024; IBM Newsroom, 2024), lo que evidencia la necesidad de soluciones que garanticen precisión, continuidad y resiliencia en sistemas basados en datos. El sistema desarrollado monitorea de forma continua los flujos de entrada de los modelos en producción; cuando identifica desviaciones estadísticamente significativas en las distribuciones de los datos (mediante pruebas Kolmogorov–Smirnov, χ^2 y PSI), activa un *pipeline* automatizado que reentrena, valida y despliega una nueva versión del modelo. Este ciclo garantiza modelos actualizados,

trazables y auditables, capaces de mantener su exactitud predictiva en condiciones de cambio. La arquitectura, basada en prácticas MLOps, integra tecnologías abiertas como Docker para contenerización, Spark para procesamiento distribuido, Jenkins para orquestación CI/CD y MLflow para versionado y trazabilidad del ciclo de vida del modelo. Los experimentos realizados en escenarios con y sin *drift* muestran que el sistema detecta las desviaciones con latencias operativas subminuto, ejecuta reentrenamientos autónomos y recupera completamente el desempeño del modelo: el F1-score inicial (≈ 0.83) retorna a 1.0 tras la actualización automática, evidenciando la eficacia del esquema de detección y respuesta. Estos resultados confirman que la automatización reduce significativamente la intervención humana, mejora la estabilidad operativa y evita degradaciones prolongadas del modelo, aportando un flujo reproducible de detección $\rightarrow$ respuesta $\rightarrow$ verificación. En términos de impacto, el proyecto contribuye a los **ODS 8, 9 y 12**: impulsa eficiencia y productividad mediante automatización inteligente (ODS 8), fortalece la infraestructura analítica y la innovación digital (ODS 9), y promueve la sostenibilidad operativa al reducir retrabajos, riesgos y desperdicio computacional (ODS 12).

Agradecimientos

A Dios, quien me dio la cotidianidad y el privilegio de sentarme cada día a aprender algo nuevo de su creación.

A mi familia, que aunque no comprendía del todo lo que hacía, su amor y confianza me impulsaron a superarme cada día.

A mis amigos, quienes quizás no conocen el contenido de este trabajo de grado, pero sí escucharon todas las quejas que surgieron durante su construcción. Mi hogar siempre estará donde ellos estén.

A mis compañeros de trabajo, presentes y pasados; porque cada una de sus enseñanzas fueron plasmadas en este documento.

A mis colegas de la Maestría en Ingeniería de Software; por convertir cada una de mis visitas en un recuerdo que no olvidaré.

A quienes ya no están cerca de mí; no solo en espíritu, sino también en distancia —sea en Valencia, Buenos Aires o Ibagué. Las noches en que fui alcanzado por recuerdos del pasado también se convirtieron en noches de inspiración para escribir las páginas que siguen.

A mí mismo, por transformar los últimos años de soledad y autocrítica en un camino del que hoy puedo sentirme orgulloso.

Y, por último, al lector que consideró este trabajo digno de ser leído; y aún más la sección de agradecimientos.

Resumen

El presente proyecto propone un sistema automatizado de reentrenamiento de modelos de *machine learning* que responde a la detección temprana del *data drift* en entornos Big Data. En la actualidad, las organizaciones enfrentan pérdidas promedio de 12,9 millones de USD anuales por mala calidad de datos y 4,88 millones por brechas de información (Gartner Inc., 2024; IBM Newsroom, 2024), lo que evidencia la necesidad de soluciones que garanticen precisión, continuidad y resiliencia en sistemas basados en datos.

El sistema desarrollado monitorea de forma continua los flujos de entrada de los modelos en producción; cuando identifica desviaciones estadísticamente significativas en las distribuciones de los datos (mediante pruebas Kolmogorov–Smirnov, χ^2 y PSI), activa un *pipeline* automatizado que reentrena, valida y despliega una nueva versión del modelo. Este ciclo garantiza modelos actualizados, trazables y auditables, capaces de mantener su exactitud predictiva en condiciones de cambio.

La arquitectura, basada en prácticas MLOps, integra tecnologías abiertas como Docker para contenerización, Spark para procesamiento distribuido, Jenkins para orquestación CI/CD y MLflow para versionado y trazabilidad del ciclo de vida del modelo. Los experimentos realizados en escenarios con y sin *drift* muestran que el sistema detecta las desviaciones con latencias operativas subminuto, ejecuta reentrenamientos autónomos y recupera completamente el desempeño del modelo: el F1-score inicial (≈ 0.83) retorna a 1.0 tras la actualización automática, evidenciando la eficacia del esquema de detección y respuesta.

Estos resultados confirman que la automatización reduce significativamente la intervención humana, mejora la estabilidad operativa y evita degradaciones prolongadas del modelo, aportando un flujo reproducible de detección $\rightarrow$ respuesta $\rightarrow$ verificación. En términos de impacto, el proyecto contribuye a los **ODS 8, 9 y 12**: impulsa eficiencia y productividad mediante automatización inteligente (ODS 8), fortalece la infraestructura analítica y la innovación digital (ODS 9), y promueve la sostenibilidad operativa al reducir retrabajos, riesgos y desperdicio computacional (ODS 12).

Palabras Clave: Data Drift, MLOps, Big Data, Automatización, Machine Learning

Abstract

This project presents an automated retraining system for machine-learning models based on early detection of data drift in Big Data environments. Organisations currently face average annual losses of USD 12.9 million due to poor data quality and USD 4.88 million due to data breaches (Gartner Inc., 2024; IBM Newsroom, 2024), underscoring the need for resilient, self-correcting data-driven systems.

The proposed system continuously monitors production data streams and, upon detecting statistically significant distributional shifts—via Kolmogorov–Smirnov, χ^2, and PSI tests—automatically triggers a retraining pipeline that validates and deploys an updated model. This ensures traceable, auditable model versions capable of maintaining predictive accuracy under changing conditions.

Following MLOps principles, the architecture integrates open-source technologies: Docker for containerisation, Spark for distributed processing, Jenkins for CI/CD orchestration, and MLflow for experiment tracking and lifecycle versioning. Experimental evaluation across drift and no-drift scenarios shows that the system achieves sub-minute detection latency, fully restores model performance, and executes end-to-end retraining without human intervention. The baseline F1-score (≈ 0.83) consistently recovers to 1.0 after automated updating, demonstrating the effectiveness of the detection–response loop.

These findings confirm that the system reduces operational burden, improves stability, and prevents prolonged performance degradation through a reproducible detect $\rightarrow$ respond $\rightarrow$ verify cycle. The project also contributes to **SDGs 8, 9, and 12** by enhancing productivity through intelligent automation (SDG 8), strengthening digital innovation and analytical infrastructure (SDG 9), and promoting sustainable operations by reducing rework, risk, and computational waste (SDG 12).

Keywords: Data Drift; MLOps; Big Data; Automated Retraining; Machine Learning

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El crecimiento exponencial en la generación de datos ha consolidado al aprendizaje automático como herramienta clave para la toma de decisiones. No obstante, los modelos no son estáticos: su rendimiento se degrada cuando cambia la distribución de entrada, fenómeno conocido como *data drift* (Sculley et al., 2015), lo que compromete la confiabilidad de sistemas inteligentes y eleva la deuda técnica.

A pesar de los avances en MLOps, persiste una brecha entre la detección estadística del *drift* y la capacidad de reaccionar automáticamente con trazabilidad completa. Muchos despliegues dependen de alertas manuales, pipelines poco reproducibles o herramientas propietarias difíciles de auditar, lo que limita la adopción de prácticas operativas consistentes en contextos latinoamericanos y de código abierto.

Esta investigación cierra dicha brecha mediante la implementación de un sistema integrado que combina monitoreo continuo, pruebas estadísticas (KS, χ^2, PSI), orquestación CI/CD y registro de artefactos en un ciclo MLOps reproducible. El pipeline —basado en Jenkins, MLflow, Spark y Prometheus— observa los datos en tiempo real, detecta desviaciones y gatilla automáticamente el reentrenamiento y despliegue del modelo, manteniendo trazabilidad y control de versiones.

La solución se valida en un entorno simulado que replica condiciones operativas (escenarios con y sin *drift*), donde los flujos de entrada generan evidencias cuantitativas sobre detección, latencia y recuperación del desempeño. El conjunto de datos empleado es **sintético** y representa **transacciones bancarias etiquetadas como fraude/no-fraude** para un ejercicio controlado de detección de movimientos fraudulentos; no se utilizan datos reales ni personales. Este cierre metodológico demuestra la factibilidad de un enfoque abierto que reduce la intervención manual y aporta lineamientos técnicos para operar modelos adaptativos en producción.

1.1. Definición del problema

1.1.1. Planteamiento del problema

En la literatura académica persiste una brecha entre las **pruebas estadísticas de drift** y su **integración operativa** en pipelines MLOps reproducibles: los estudios reportan $KS/\chi^2/PSI$ u otros detectores, pero rara vez documentan cómo se enlazan con orquestación CI/CD abierta, políticas anti-*flapping* o trazabilidad completa. Este vacío teórico-práctico es especialmente notorio en entornos Big Data abiertos, donde no existen benchmarks estandarizados ni referencias arquitectónicas auditables. Sobre esta brecha técnica se sustenta la justificación económica posterior: sin un pipeline integrado y comprobable, los costos por degradación del modelo y la deuda técnica continúan escalando.

En el contexto global actual, caracterizado por una explosión de datos sin precedentes, se estima que el *Global DataSphere* alcanzará los 181 Zettabytes en 2025 (Bartley, 2024). Esta expansión está impulsada por el crecimiento acelerado de tecnologías como la inteligencia artificial generativa, el Internet de las Cosas (IoT), y la digitalización masiva de procesos. Sin embargo, la utilidad real de estos datos se ve comprometida por problemas de calidad y gobernanza. Según Gartner, la mala calidad de los datos le cuesta a las organizaciones, en promedio, 12,9 millones de dólares anuales (Gartner Inc., 2024), mientras que el costo promedio por brecha de datos se sitúa en 4,88 millones de dólares (IBM Newsroom, 2024).

En este escenario, el uso de modelos de machine learning se ha generalizado como herramienta para la toma de decisiones automatizadas. No obstante, estudios recientes revelan que cerca del 91 % de los modelos desplegados en producción sufren algún tipo de *model drift* en su primer año de operación (Dilmegani, 2025; Breck et al., 2017). Este fenómeno, conocido como *data drift* cuando afecta la distribución de entrada de los datos, degrada el rendimiento predictivo, comprometiendo decisiones operativas, regulatorias y comerciales.

Para mantener la fluidez narrativa sin sacrificar evidencia, los principales indicadores cuantitativos se sintetizan en la Tabla 1.1, mientras que los detalles y series temporales se documentan en el Anexo A. La tabla resume tanto el crecimiento global de los datos como las brechas regionales de madurez y calidad, proporcionando el punto de partida para el problema técnico abordado en esta tesis.

Cuadro 1.1: Indicadores contextuales sobre data drift y madurez MLOps.

Indicador	**Valor / Fuente**
Explosión de datos globales	*Global DataSphere* proyectado en 181 ZB para 2025 (Bartley, 2024).
Costo de mala calidad de datos	Promedio global de USD 12.9 M por organización al año (Gartner Inc., 2024).
Incidencia de *model drift*	91 % de los modelos sufren degradación en el primer año (Dilmegani, 2025; Breck et al., 2017).
Inversión en centros de datos LATAM	USD 6.36 B (2023) con CAGR 7.95 % hasta 2029 (Helmi Group, 2024).
Calidad de datos en Colombia	56 % de bases corporativas con deficiencias críticas (Deyde DataCentric, 2023).
Inversión pública en IA	COP 480 mil millones para infraestructura/capacitación (Política IA 2025) (Consejo Nacional de Política Económica y Social (CONPES), 2025).
Impacto económico del drift	Pérdida > USD 12 M por caída de 3 pp en precisión de modelos crediticios (Kim et al., 2018).

A nivel regional, la inversión tecnológica no se ha traducido en la madurez operativa necesaria para sostener modelos en producción; persisten retos de calidad del dato, cultura organizacional y

adopción de prácticas MLOps (Helmi Group, 2024; Deyde DataCentric, 2023). Incluso iniciativas públicas como la Política Nacional de IA (2025) carecen de lineamientos concretos para pipelines automatizados que mitiguen el *drift* (Consejo Nacional de Política Económica y Social (CONPES), 2025). Desde el punto de vista económico, la ausencia de mecanismos automatizados se refleja en pérdidas millonarias y en decisiones subóptimas en sectores regulados (Kim et al., 2018). Estas brechas justifican la necesidad de soluciones reproducibles que integren detección, reentrenamiento y trazabilidad.

En el plano social y de desarrollo sostenible, esta problemática impacta la eficiencia de los servicios digitales, la equidad tecnológica y la sostenibilidad organizacional. Su vínculo con los Objetivos de Desarrollo Sostenible (ODS) es directo:

- **ODS 8 – Trabajo decente y crecimiento económico:** La degradación en el rendimiento de los modelos compromete la productividad de procesos automatizados, disminuye la competitividad y obstaculiza la generación de empleo calificado en áreas tecnológicas. Un sistema automatizado de actualización de modelos puede contribuir a mantener la eficiencia operativa y abrir nuevas oportunidades de innovación.
- **ODS 9 – Industria, innovación e infraestructura:** El uso ineficiente de modelos obsoletos limita la innovación tecnológica y reduce la efectividad de los procesos industriales basados en datos. La automatización del reentrenamiento fortalece la infraestructura analítica y promueve una cultura de mejora continua.
- **ODS 12 – Producción y consumo responsables:** Decisiones mal informadas debido a modelos desactualizados pueden conllevar a sobrecostos, desperdicio de recursos y fallas logísticas. Automatizar la actualización de modelos con base en datos actuales permite una asignación más eficiente y responsable de los recursos.

Desde el punto de vista técnico, las soluciones actuales presentan limitaciones significativas. La mayoría de las organizaciones aún dependen de monitoreo reactivo en paneles aislados, sin mecanismos de detección y respuesta automatizados. Los pipelines de reentrenamiento suelen ser ad-hoc, sin adherencia a prácticas de integración y despliegue continuo (CI/CD), y carecen de capacidades para orquestación multinube, contenerización uniforme y cumplimiento regulatorio.

En síntesis, aunque la literatura ha documentado múltiples enfoques de detección de data drift y mecanismos de reentrenamiento, persisten vacíos en torno a su integración práctica en pipelines completamente automatizados y reproducibles bajo esquemas CI/CD y observabilidad de extremo a extremo. La mayoría de propuestas carece de validación empírica en entornos Big Data o de estrategias abiertas que permitan su réplica. Este proyecto busca aportar una contribución metodológica concreta al desarrollar y documentar una arquitectura open-source reproducible que combine detección estadística de drift, automatización CI/CD y monitoreo operacional completo, constituyendo un avance práctico frente a ese rezago de la literatura.

1.1.2. Formulación del problema

Pregunta central. *¿Cómo diseñar, desarrollar e implementar un sistema productivo robusto basado en un pipeline MLOps, que detecte automáticamente* data drift*, active reentrenamiento y despliegue continuo, minimizando intervención humana, costos operativos y garantizando alta disponibilidad en entornos reales Big Data?*

Enunciado de validación del enfoque propuesto.

> Se plantea que un sistema productivo basado en MLOps, con capacidades de detección automática de *data drift*, monitoreo continuo y reentrenamiento automatizado, puede mejorar sustancialmente la estabilidad operativa y la precisión de los modelos de *machine learning* al reducir la intervención manual y facilitar la actualización continua en entornos de datos dinámicos.

Aspectos clave a demostrar durante el desarrollo del proyecto.

1. Que los mecanismos automáticos de alerta y análisis estadístico permiten identificar desviaciones en la distribución de datos que afecten el rendimiento del modelo.
2. Que la automatización del reentrenamiento reduce la necesidad de intervención humana y mejora la continuidad operativa del sistema.
3. Que la trazabilidad y gestión del ciclo de vida de los modelos facilita su mantenimiento, validación y auditoría en producción.
4. Que una arquitectura integrada y replicable permite desplegar flujos MLOps en contextos reales que requieren adaptabilidad frente al cambio de datos.

1.2. Objetivos del proyecto

1.2.1. Objetivo General

Construir un sistema automatizado de reentrenamiento que, mediante la detección temprana de *data drift*, active un ciclo MLOps en entornos Big Data, asegurando la actualización continua y el reentrenamiento de los modelos de machine learning.

1.2.2. Objetivos específicos

- Implementar una infraestructura en la nube escalable que permita el procesamiento eficiente de grandes volúmenes de datos, integrando mecanismos de monitorización y evaluación continua para identificar y responder oportunamente al *data drift*.

- Desarrollar un sistema computacional que combine métodos estadísticos y algoritmos supervisados para la detección automática de *data drift* en tiempo real, generando alertas que activen procesos automatizados de reentrenamiento sin intervención humana.
- Validar en un escenario simulado con y sin *data drift*, la eficacia del sistema computacional tras la detección de desviaciones, midiendo precisión, recall y latencia para asegurar la actualización continua y la eficiencia operativa.

1.3. Delimitaciones y alcances

El proyecto se concentró en la evaluación de un prototipo funcional basado en una arquitectura de Big Data previamente documentada en la literatura, evitando el diseño de una nueva infraestructura desde cero. Se utilizará como referencia una configuración de clúster pseudo-distribuido que emplea contenedores Docker y Docker Compose, replicando un sistema similar a HDFS y herramientas de procesamiento de datos como Apache Spark en un entorno controlado. Este montaje es, por tanto, un **entorno Big Data simulado** que permite observar el comportamiento del pipeline con control experimental absoluto; aunque no se ejecuta sobre un clúster físico multi-nodo, la arquitectura (Spark, HDFS, Jenkins, MLflow) está declarada y parametrizada para escalar en entornos distribuidos reales con mínimos ajustes de infraestructura (p. ej., despliegues en Kubernetes o servicios administrados).

Además, se adoptó la arquitectura propuesta por (Chen et al., 2022), la cual integra un sistema de almacenamiento distribuido con capacidades de procesamiento en tiempo real y batch, facilitando el manejo de grandes volúmenes de información. Esta arquitectura se complementa con sistemas de monitorización y alertas automáticas que permiten la detección de *data drift* y, consecuentemente, la activación de *pipelines* automáticos de reentrenamiento.

Para la validación del sistema, se desarrolló dos escenarios de prueba mediante la generación de datos semi-aleatorios utilizando la biblioteca `Faker` de Python. El generador emula **transacciones bancarias con etiquetas de fraude/no-fraude** (monto, tipo de cuenta, canal, `risk_score`) para simular el caso de uso de detección de movimientos fraudulentos sin exponer información real. El primer escenario sirve como base para el entrenamiento inicial del modelo de *machine learning*, mientras que en el segundo se introduce una variación controlada en la distribución de los datos (*data drift* intencional) para demostrar el funcionamiento del *pipeline* de reentrenamiento automático. Se implementarán mecanismos de detección de *data drift* mediante pruebas estadísticas como Kolmogorov-Smirnov, Chi-cuadrado y la divergencia de Kullback-Leibler, permitiendo evaluar el impacto en las métricas del modelo. La implementación del *pipeline* automatizado mediante Jenkins se concentrará en un modelo de *machine learning* de prueba, permitiendo evaluar la eficacia del sistema sin extender el análisis a múltiples modelos en paralelo. La integración de MLflow facilitará el registro y comparación de cada ciclo de reentrenamiento, mientras que *dashboards* en Grafana ofrecerán una visualización en tiempo real de métricas como precisión, *recall*, F1-score y tiempos de respuesta del sistema. En otros dominios (por ejemplo, series de demanda o mantenimiento predictivo), el mismo flujo requeriría sustituir el generador por variables representativas del dominio y

ajustar las métricas de validación, pero la lógica de monitoreo KS/χ^2/PSI y el disparo de Jenkins permanecerían invariantes.

Finalmente, aunque el prototipo se implementa en un entorno local con Docker y Docker Compose, la arquitectura propuesta es **portátil hacia la nube**, pudiendo migrarse de manera sencilla a plataformas como **Azure Machine Learning**. Esto permitiría aprovechar clústeres administrados de cómputo, escalado automático de recursos y la integración con **Azure Monitor** para fortalecer las capacidades de observabilidad y operación continua del sistema.

Tipo de *drift* evaluado. La validación empírica se concentra en **covariate drift** (alteraciones de las distribuciones de entrada) controlado sobre las variables financieras y de riesgo. La modificación del `risk_score` introduce sólo una aproximación parcial al **concept drift**; no se manipula directamente la etiqueta ni se simula un cambio estructural en la función objetivo. En consecuencia, los resultados demuestran sensibilidad y recuperación frente a *covariate drift* y establecen un punto de partida para futuras extensiones que aborden *concept drift* reforzado o supervisado.

1.4. Justificación del trabajo de grado

La creciente dependencia de modelos de *machine learning* para la toma de decisiones estratégicas en entornos productivos obliga a mantener altos niveles de precisión y rendimiento. Sin embargo, cambios continuos en la distribución de los datos, conocidos como *data drift*, generan degradaciones críticas en la precisión de los modelos, incrementando costos operativos y riesgos regulatorios cuando no son detectados y gestionados oportunamente.

Este proyecto justifica su relevancia al proponer un producto tecnológico innovador que automatiza la detección temprana y el reentrenamiento continuo de modelos en producción. Al integrar herramientas avanzadas de monitorización y técnicas estadísticas robustas con pipelines automatizados mediante metodologías MLOps, la solución reduce significativamente la intervención humana, minimiza errores operativos y mejora la rapidez en la respuesta ante desviaciones.

Adicionalmente, la propuesta establece un estándar metodológico replicable, facilitando la adopción de prácticas ágiles y seguras en el ciclo completo de vida de los modelos. De esta forma, el sistema propuesto no solo asegura una alta precisión predictiva y estabilidad operativa, sino que también impulsa la innovación y competitividad de las organizaciones que gestionan grandes volúmenes de datos dinámicos (Kim et al., 2018; Merkel, 2014; Breck et al., 2017; Gama et al., 2014).

1.5. Metodología de la investigación

La metodología se presenta aquí de forma sintética para evitar duplicidad con los Capítulos 3 y 4. El estudio es **aplicado y cuasi-experimental**: valida un pipeline MLOps que *detecta data drift* y *reentrena* automáticamente.

Diseño general. Dos escenarios: E1 (sin *drift*) y E2 (con *drift* inducido). Se observan métricas de desempeño (F1, Recall, AUC), latencias operativas (*TTFD*, *TTR*) y magnitud de cambio (*PSI*). Las políticas de activación/cooldown, el detalle de hipótesis, variables y análisis estadístico se desarrollan en el Capítulo 4.1 y 4.1.3.

Implementación. La infraestructura y componentes técnicos (Spark/HDFS, Jenkins, MLflow, Prometheus/Grafana, Docker Compose) se documentan en el Capítulo 3.1. Aquí solo se describe la lógica metodológica y el alcance experimental; los detalles operativos quedan en los capítulos respectivos.

1.5.1. Fases de desarrollo e implementación

La investigación adoptó una **metodología aplicada y experimental**, organizada en fases secuenciales que abarcan desde el diseño conceptual hasta la validación empírica del sistema. El propósito fue construir y evaluar un *pipeline* de reentrenamiento automatizado basado en detección temprana de *data drift*.

Cada fase combinó **principios de ingeniería reproducible** —contenedorización, trazabilidad y modularidad— con **criterios de validación cuantitativa** orientados a métricas de desempeño y eficiencia operativa. El enfoque técnico se apoyó en herramientas de código abierto (Python, Spark, Jenkins, MLflow y Prometheus), desplegadas mediante **Docker Compose** para garantizar replicabilidad y control experimental.
La Figura 1.6 sintetiza el flujo metodológico del proyecto, integrando las etapas de datos, automatización y observabilidad dentro de un ciclo continuo de mejora. Este diagrama no representa la implementación detallada, sino la **lógica de investigación** que guía la transición desde el diseño hasta la validación del sistema.

Síntesis metodológica.

- **Enfoque cuantitativo/cuasi-experimental:** se contrastan escenarios con/ sin drift (E1/E2) y se evalúa la recuperación post-reentrenamiento (E2 post) mediante métricas F1, AUC, TTFD y TTR.
- **Variables principales:** presencia de drift (independiente), F1/Recall/PSI/TTFD/TTR (dependientes) y parámetros de infraestructura como controles (ventanas, semillas, umbrales).
- **Diseño de hipótesis:** H_0 (sin mejora tras reentrenamiento) vs. H_1 (pipeline automatizado mejora desempeño y latencia frente al drift).

Elección de pruebas estadísticas. Se usan **KS**, χ^2 y **PSI** porque cubren los tres casos críticos para monitoreo de datos: variables numéricas con distribuciones no asumidas (**KS** es no paramétrica y sensible a cambios de forma/media); variables categóricas con cambios en proporciones (χ^2 contrasta frecuencias esperadas vs. observadas); y un indicador agregado, de interpretación operativa, sobre el *score* del modelo (**PSI**, ampliamente usado en riesgo crediticio pero aplicable a cualquier dominio que requiera vigilar estabilidad de salidas/atributos con *binning* explícito). Esta combinación equilibra sensibilidad y robustez sin exigir supuestos fuertes de normalidad o varianza, y permite explicar alertas a audiencias técnicas y de negocio; los umbrales PSI deben calibrarse por dominio (p. ej., 0.1/0.2/0.3 en riesgo, valores más bajos en contextos muy volátiles).

Multiplicidad, ventanas y decisión de alcance. El uso simultáneo de KS/χ^2/PSI con múltiples columnas implica corrección por multiplicidad (Holm/BH) y calibración de tamaños de ventana para evitar falsos positivos con muestras pequeñas; esta tesis adopta ventanas fijas (1000 filas, 30 s) y regla OR con $\alpha = 0.01$, delegando la exploración multivariada (MMD, Energy) a trabajo futuro por su mayor costo y menor trazabilidad en este prototipo.

Reproducibilidad operativa. Las corridas experimentales se pueden replicar con los scripts versionados en `./scripts` y `./metrics_test`, usando las semillas fijadas (`seed`=42,7) y los DAGs `bank_data_generation_dag` / `bank_data_generation_and_ml`; los parámetros de entorno (`.env`) documentan umbrales y rutas HDFS/Spark, y MLflow conserva artefactos y métricas por réplica.

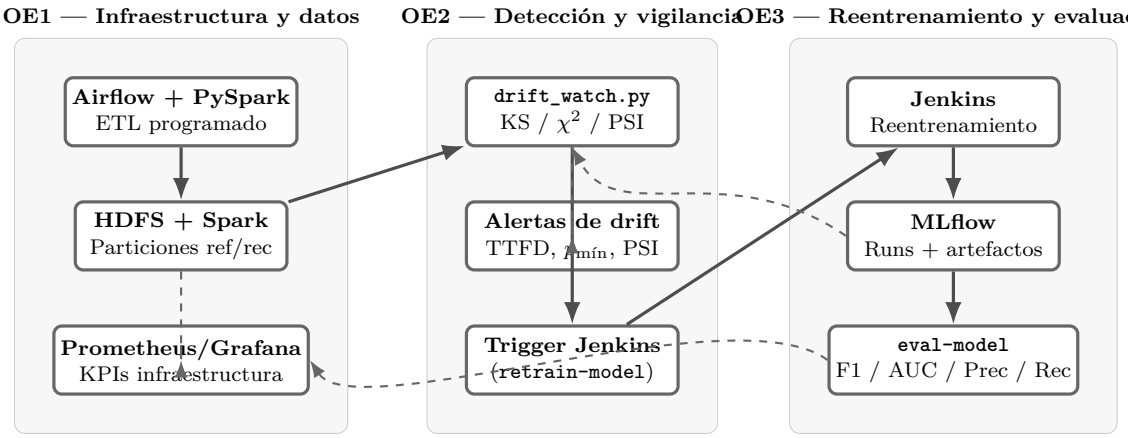


Figura 1.1: Diagrama integrado de fases: OE1 despliega la infraestructura y la ingesta; OE2 introduce la vigilancia estadística y los *triggers*; OE3 automatiza el reentrenamiento y evalúa F1/AUC.

A continuación, se describen los pasos metodológicos principales:

1. **Revisión de literatura y diseño conceptual:** En esta fase se realizó una exploración bibliográfica intensiva para establecer las bases teóricas y prácticas del proyecto. Se estudiaron técnicas de detección de *data drift* (como Kolmogorov-Smirnov, Chi-cuadrado y Kullback-Leibler), arquitecturas para entornos Big Data y principios de MLOps aplicados a flujos de trabajo de actualización de modelos. Este análisis permitió definir los módulos funcionales que compusieron el sistema, identificar tecnologías viables y proponer una arquitectura conceptual sobre la cual se desarrolló el prototipo. Se consolidaron los principales patrones arquitectónicos y se bosquejó un primer diseño lógico.

   La documentación generada incluye el registro de papers clave, anotaciones de arquitectura, bitácoras de decisiones y estructuras base de directorios en Git.

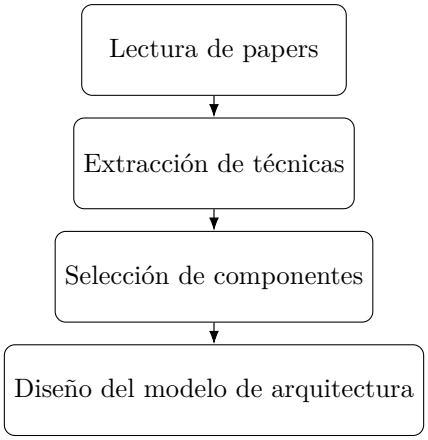


Figura 1.2: Diagrama: Revisión y diseño conceptual

2. **Implementación del entorno de prueba:** Se habilitó un entorno experimental pseudo-distribuido empleando contenedores orquestados con Docker Compose. Esta estrategia permitió aislar cada componente del sistema, controlar versiones, replicar la arquitectura en distintos entornos y escalar los servicios de manera controlada, reproduciendo condiciones similares a las de un sistema en producción.

   En el primer paso, el uso conjunto de **Docker + Compose** facilitarón el despliegue modular de todos los servicios involucrados: desde la gestión de datos hasta el procesamiento y monitoreo. Docker Compose coordinará múltiples contenedores definidos en un archivo de configuración, permitiendo instanciar la infraestructura de forma declarativa, eficiente y reproducible.

   A continuación, se puso en ejecución un **contenedor con Apache Spark y HDFS**. Spark funcionará como el motor de procesamiento distribuido sobre el cual se ejecutó transformaciones, agregaciones y simulaciones en los datos, validando así el comportamiento del sistema bajo diferentes volúmenes y velocidades. HDFS proporcionará almacenamiento persistente, imitando entornos reales donde los datos no son efímeros y deben estar disponibles para procesos posteriores como reentrenamiento o análisis de rendimiento.

   Luego, se implementó un módulo en Python responsable del **procesamiento de datos sintéticos y su transformación**. Aunque los datos no se generarán en esta fase, sí se transformarán para representar distintos tipos de alteraciones en la distribución (por ejemplo, desplazamientos, cambios en la varianza o presencia de valores atípicos). Estas transformaciones sirvieron para alimentar los flujos que simulen el **data drift**.

   Finalmente, se definieron un conjunto de **escenarios controlados** que permitan medir con precisión la capacidad del sistema para reaccionar ante los eventos simulados. Estos escenarios fueron diseñados para comparar el rendimiento del pipeline con y sin automatización, permitiendo evaluar la latencia en la detección, la estabilidad del modelo tras el reentrenamiento, y la trazabilidad completa de los cambios inducidos.

Se documentaron los archivos de configuración YAML de Docker Compose, scripts de inicialización, métricas base y escenarios de validación, junto con anotaciones en el repositorio.

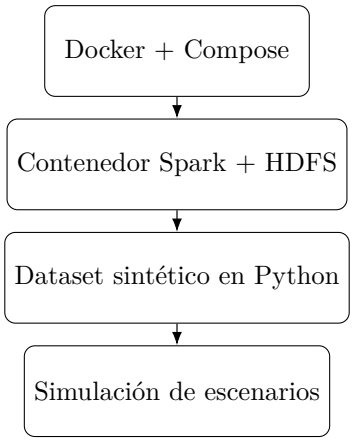


Figura 1.3: Diagrama: Implementación del entorno de prueba

3. **Desarrollo del pipeline de detección y reentrenamiento:** Esta fase contempla la construcción del núcleo funcional del sistema: un pipeline automatizado que detecta desviaciones en la distribución de los datos (*data drift*) y activa mecanismos de reentrenamiento del modelo en producción. La arquitectura del pipeline está orientada a la operación continua y a la capacidad de respuesta autónoma ante cambios significativos en los datos de entrada.

4. **Uso de Airflow como ETL:** Aunque la prueba inicial se realizó ejecutando directamente los scripts de generación de particiones en HDFS, el diseño metodológico incorpora **Apache Airflow** como orquestador del proceso ETL. Para ello se implementó un **DAG** (`bank_data_generation_dag`) que ejecuta periódicamente un contenedor Docker con el cliente PySpark (`arlequin-pyspark-client`). Este DAG invoca el script `generate_data.py`, que sintetiza transacciones bancarias etiquetadas (fraude/no-fraude) con probabilidad de *drift* y las escribe en HDFS en formato Parquet.

   Representación del DAG de Airflow El flujo ETL se implementó como un *Directed Acyclic Graph (DAG)* en Airflow que orquesta la generación, transformación y carga de datos sintéticos hacia HDFS. Cada nodo corresponde a una tarea contenedorizada (Docker) ejecutada por el cliente PySpark.

   (Se omite pseudocódigo/diagrama extensos del DAG para reducir extensión; el flujo completo está en el repositorio y anexos técnicos).

   El flujo del DAG contempla:

   a) **Extracción**: inicialización de un lote de datos sintéticos mediante `Faker` y reglas de estacionalidad.

b) **Transformación**: creación de un `DataFrame` en Spark con esquema predefinido, incorporación de campos derivados (p.ej. `risk_score`) y aplicación del factor de drift.

c) **Carga**: escritura en `HDFS` bajo la ruta `hdfs:///user/bank_data/bank_transactions`, con particionado temporal (`timestamp`) para habilitar ingestas incrementales y pruebas de detección de *drift*.

La inclusión de Airflow aporta **reproducibilidad y trazabilidad** (cada corrida del DAG queda registrada), **aislamiento de responsabilidades** (Airflow únicamente orquesta la generación/carga, mientras el *drift-watcher* monitorea particiones nuevas) y **operación continua** (el intervalo de scheduling del DAG determina el ritmo de llegada de datos y, por ende, la frecuencia de evaluación del *drift*).

5. **Entrada del sistema:** el pipeline procesará flujos de datos provenientes del entorno experimental configurado previamente, almacenados en el sistema distribuido y transformados por procesos previos de limpieza y estructuración.

6. **Evaluación estadística:** la detección de drift se implementó mediante una combinación de pruebas estadísticas no paramétricas sobre ventanas móviles de datos. Se utilizaron:

   - **Kolmogorov-Smirnov (KS):** para comparar la distribución empírica de nuevas observaciones con la distribución histórica del entrenamiento.
   - **Chi-cuadrado (χ^2):** para detectar cambios discretos en distribuciones categóricas.
   - **Kullback-Leibler Divergence (KL):** para evaluar la diferencia entre distribuciones de probabilidad observadas y esperadas.

   Estas pruebas se ejecutaron usando librerías de Python como `scipy.stats` y `alibi-detect`. Se configuraron umbrales adaptativos que disparen alertas cuando se supere un nivel crítico de desviación.

7. **Mecanismo de decisión y activación:** cuando las pruebas detecten *data drift*, se activó una tarea automatizada orquestada por `Jenkins`. Esta tarea lanzó un nuevo proceso de reentrenamiento con los datos recientes. Este componente incluyó validación cruzada y control de sobreajuste mediante técnicas como `early stopping` y registro de métricas.

8. **Reentrenamiento y versionado:** una vez generado el nuevo modelo, se almacenó junto con sus métricas, hiperparámetros y configuración en `MLflow`. Esto garantizará la trazabilidad completa del ciclo de vida del modelo, incluyendo comparaciones con versiones anteriores para validar mejoras y evitar regresiones.

9. **Salida esperada:** un modelo actualizado validado bajo condiciones de *drift*, métricas registradas y documentadas, y control de versiones centralizado.

   Se documentó cada versión del modelo, scripts de validación, configuraciones de ejecución de Jenkins y resultados comparativos.

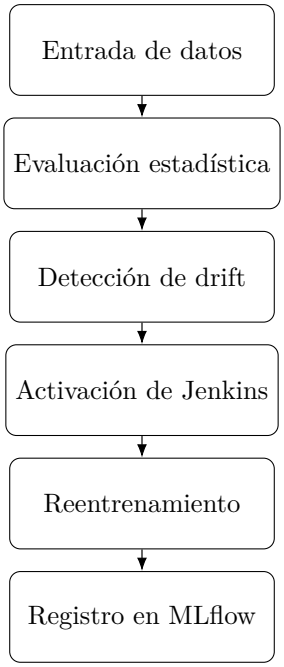


Figura 1.4: Diagrama: Pipeline de detección y reentrenamiento

10. **Validación del sistema:** La fase de validación constituye un componente esencial del proyecto, pues permite determinar en qué medida el sistema implementado cumple con los objetivos planteados de detección temprana de *data drift* y reentrenamiento automatizado. Para ello se diseñaron y ejecutaron dos escenarios controlados de prueba: (i) un escenario estático sin presencia de *drift*, que servió como línea base de comparación, y (ii) un escenario con *drift* inducido de manera progresiva mediante la alteración controlada de variables de entrada, a fin de evaluar la capacidad del sistema para detectar desviaciones y restaurar el desempeño del modelo.

    En cada escenario se registraron métricas de desempeño del modelo, incluyendo precisión, *recall*, F1-score y AUC, complementadas con indicadores de operación del pipeline como la latencia de detección, el tiempo total de reentrenamiento y los recursos computacionales consumidos (CPU, memoria y uso de disco). Estas métricas se capturaron de manera automática a través de MLflow para el versionamiento de modelos, Prometheus para la exposición de indicadores en tiempo real, y Grafana para la visualización y consolidación de paneles comparativos.

    El proceso experimental considerará además la generación de múltiples corridas bajo condiciones equivalentes, con el fin de asegurar reproducibilidad y obtener promedios estadísticamente significativos. De esta manera, se podrá distinguir entre fluctuaciones aleatorias y comportamientos sistemáticos del sistema frente al *data drift*. Asimismo, se documentaron aspectos de eficiencia operativa, tales como el consumo de recursos durante el reentrenamiento, el impacto

del tamaño de las particiones de datos en la detección del *drift*, y el efecto de los umbrales de significancia establecidos para activar el pipeline.

El análisis de resultados no se limitará únicamente a la comparación numérica de métricas, sino que incluyó la elaboración de informes interpretativos por versión del modelo. Estos informes contendrán gráficos de evolución de métricas, tablas comparativas de desempeño entre escenarios y descripciones cualitativas de la respuesta del sistema. Además, se elaborarán visualizaciones específicas para resaltar la relación entre el *Population Stability Index* (PSI) y las métricas de rendimiento, con el propósito de evaluar el poder predictivo del PSI como señal temprana de degradación del modelo.

Esta fase producirá como entregables un conjunto de informes comparativos, visualizaciones interactivas en dashboards, métricas consolidadas y resúmenes analíticos que permitirán evaluar de manera integral la eficacia del sistema. Los resultados obtenidos sirvieron como insumo directo para la discusión final y la formulación de recomendaciones para futuros despliegues en entornos reales.

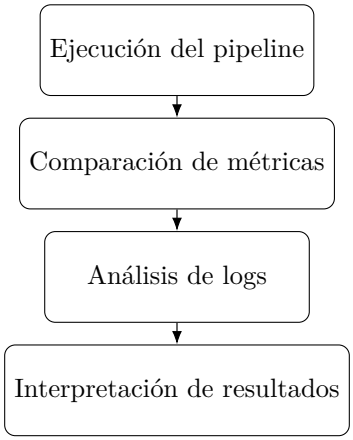


Figura 1.5: Diagrama: Validación del sistema

1.5.2. Fases de desarrollo e implementación

La investigación adoptó una **metodología aplicada y experimental**, organizada en fases secuenciales que abarcan desde el diseño conceptual hasta la validación empírica del sistema. El propósito fue construir y evaluar un *pipeline* de reentrenamiento automatizado basado en detección temprana de *data drift*.

Cada fase combinó **principios de ingeniería reproducible** —contenedorización, trazabilidad y modularidad— con **criterios de validación cuantitativa** orientados a métricas de desempeño y eficiencia operativa. El enfoque técnico se apoyó en herramientas de código abierto (Python, Spark, Jenkins, MLflow y Prometheus), desplegadas mediante **Docker Compose** para garantizar replicabilidad y control experimental.

La Figura 1.6 sintetiza el flujo metodológico del proyecto, integrando las etapas de datos, automatización y observabilidad dentro de un ciclo continuo de mejora. Este diagrama no representa la implementación detallada, sino la **lógica de investigación** que guía la transición desde el diseño hasta la validación del sistema.

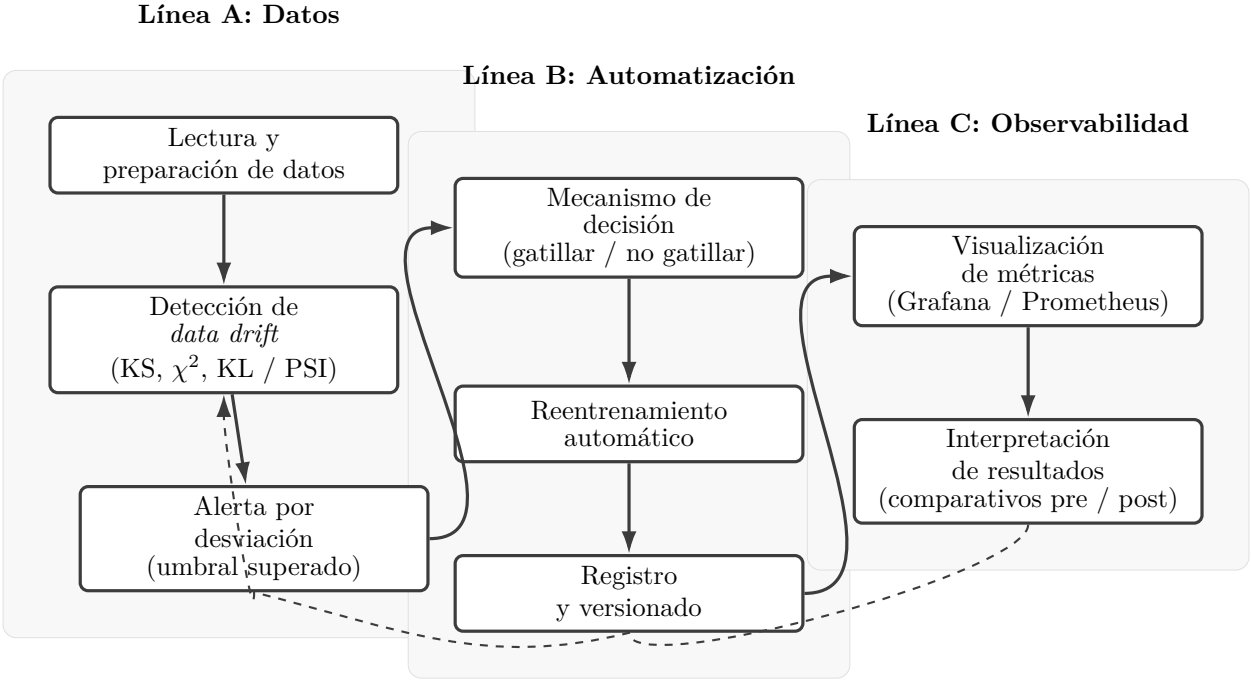


Figura 1.6: Flujo de trabajo metodológico: datos → detección → alerta → reentrenamiento → registro → visualización, con bucle de mejora continua.

A continuación, se describen los pasos metodológicos principales:

1. **Revisión de literatura y diseño conceptual:** En esta fase se realizó una exploración bibliográfica intensiva para establecer las bases teóricas y prácticas del proyecto. Se estudiaron técnicas de detección de *data drift* (como Kolmogorov-Smirnov, Chi-cuadrado y Kullback-Leibler), arquitecturas para entornos Big Data y principios de MLOps aplicados a flujos de trabajo de actualización de modelos. Este análisis permitió definir los módulos funcionales que compusieron el sistema, identificar tecnologías viables y proponer una arquitectura conceptual sobre la cual se desarrolló el prototipo. Se consolidaron los principales patrones arquitectónicos y se bosquejó un primer diseño lógico.

   La documentación generada incluye el registro de papers clave, anotaciones de arquitectura, bitácoras de decisiones y estructuras base de directorios en Git.

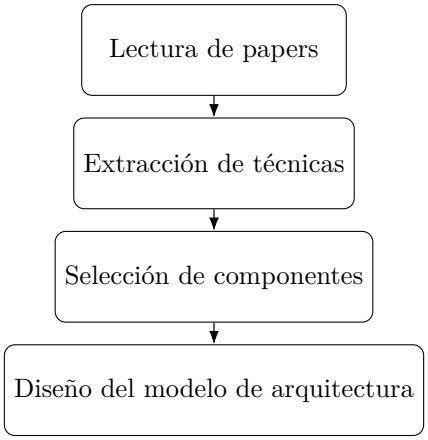


Figura 1.7: Diagrama: Revisión y diseño conceptual

2. **Implementación del entorno de prueba:** Se habilitó un entorno experimental pseudo-distribuido empleando contenedores orquestados con Docker Compose. Esta estrategia permitió aislar cada componente del sistema, controlar versiones, replicar la arquitectura en distintos entornos y escalar los servicios de manera controlada, reproduciendo condiciones similares a las de un sistema en producción.

   En el primer paso, el uso conjunto de **Docker + Compose** facilitarón el despliegue modular de todos los servicios involucrados: desde la gestión de datos hasta el procesamiento y monitoreo. Docker Compose coordinará múltiples contenedores definidos en un archivo de configuración, permitiendo instanciar la infraestructura de forma declarativa, eficiente y reproducible.

   A continuación, se puso en ejecución un **contenedor con Apache Spark y HDFS**. Spark funcionará como el motor de procesamiento distribuido sobre el cual se ejecutó transformaciones, agregaciones y simulaciones en los datos, validando así el comportamiento del sistema bajo diferentes volúmenes y velocidades. HDFS proporcionará almacenamiento persistente, imitando entornos reales donde los datos no son efímeros y deben estar disponibles para procesos posteriores como reentrenamiento o análisis de rendimiento.

   Luego, se implementó un módulo en Python responsable del **procesamiento de datos sintéticos y su transformación**. Aunque los datos no se generarán en esta fase, sí se transformarán para representar distintos tipos de alteraciones en la distribución (por ejemplo, desplazamientos, cambios en la varianza o presencia de valores atípicos). Estas transformaciones sirvieron para alimentar los flujos que simulen el **data drift**.

   Finalmente, se definieron un conjunto de **escenarios controlados** que permitan medir con precisión la capacidad del sistema para reaccionar ante los eventos simulados. Estos escenarios fueron diseñados para comparar el rendimiento del pipeline con y sin automatización, permitiendo evaluar la latencia en la detección, la estabilidad del modelo tras el reentrenamiento, y la trazabilidad completa de los cambios inducidos.

Se documentaron los archivos de configuración YAML de Docker Compose, scripts de inicialización, métricas base y escenarios de validación, junto con anotaciones en el repositorio.

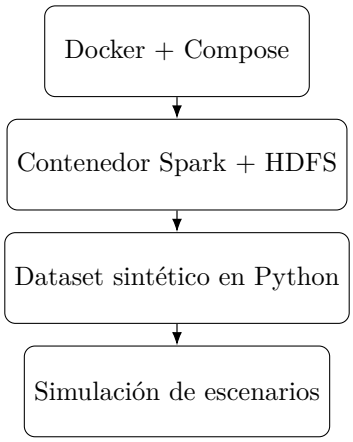


Figura 1.8: Diagrama: Implementación del entorno de prueba

3. **Desarrollo del pipeline de detección y reentrenamiento:** Esta fase contempla la construcción del núcleo funcional del sistema: un pipeline automatizado que detecta desviaciones en la distribución de los datos (*data drift*) y activa mecanismos de reentrenamiento del modelo en producción. La arquitectura del pipeline está orientada a la operación continua y a la capacidad de respuesta autónoma ante cambios significativos en los datos de entrada.

4. **Uso de Airflow como ETL:** Aunque la prueba inicial se realizó ejecutando directamente los scripts de generación de particiones en HDFS, el diseño metodológico incorpora **Apache Airflow** como orquestador del proceso ETL. Para ello se implementó un **DAG** (`bank_data_generation_dag`) que ejecuta periódicamente un contenedor Docker con el cliente PySpark (`arlequin-pyspark-client`). Este DAG invoca el script `generate_data.py`, que sintetiza transacciones bancarias etiquetadas (fraude/no-fraude) con probabilidad de *drift* y las escribe en HDFS en formato Parquet.

Representación del DAG de Airflow El flujo ETL se implementó como un *Directed Acyclic Graph (DAG)* en Airflow que orquesta la generación, transformación y carga de datos sintéticos hacia HDFS. Cada nodo corresponde a una tarea contenedorizada (Docker) ejecutada por el cliente PySpark.

El flujo del DAG contempla:

a) **Extracción**: inicialización de un lote de datos sintéticos mediante `Faker` y reglas de estacionalidad.

b) **Transformación**: creación de un `DataFrame` en Spark con esquema predefinido, incorporación de campos derivados (p.ej. `risk_score`) y aplicación del factor de drift.

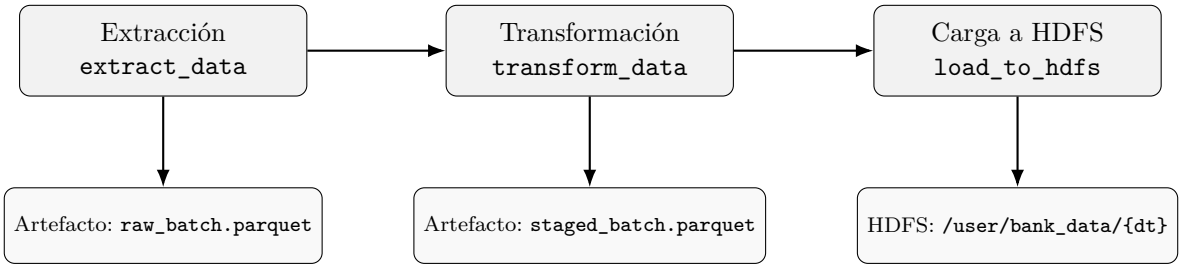


Figura 1.9: Flujo ETL orquestado por Airflow: nodos, dependencias y artefactos.

c) **Carga**: escritura en `HDFS` bajo la ruta `hdfs:///user/bank_data/bank_transactions`, con particionado temporal (`timestamp`) para habilitar ingestas incrementales y pruebas de detección de *drift*.

La inclusión de Airflow aporta **reproducibilidad y trazabilidad** (cada corrida del DAG queda registrada), **aislamiento de responsabilidades** (Airflow únicamente orquesta la generación/carga, mientras el *drift-watcher* monitorea particiones nuevas) y **operación continua** (el intervalo de scheduling del DAG determina el ritmo de llegada de datos y, por ende, la frecuencia de evaluación del *drift*).

5. **Entrada del sistema:** el pipeline procesará flujos de datos provenientes del entorno experimental configurado previamente, almacenados en el sistema distribuido y transformados por procesos previos de limpieza y estructuración.

6. **Evaluación estadística:** la detección de drift se implementó mediante una combinación de pruebas estadísticas no paramétricas sobre ventanas móviles de datos. Se utilizaron:

   - **Kolmogorov-Smirnov (KS):** para comparar la distribución empírica de nuevas observaciones con la distribución histórica del entrenamiento.
   - **Chi-cuadrado (χ^2):** para detectar cambios discretos en distribuciones categóricas.
   - **Kullback-Leibler Divergence (KL):** para evaluar la diferencia entre distribuciones de probabilidad observadas y esperadas.

   Estas pruebas se ejecutaron usando librerías de Python como `scipy.stats` y `alibi-detect`. Se configuraron umbrales adaptativos que disparen alertas cuando se supere un nivel crítico de desviación.

7. **Mecanismo de decisión y activación:** cuando las pruebas detecten *data drift*, se activó una tarea automatizada orquestada por `Jenkins`. Esta tarea lanzó un nuevo proceso de reentrenamiento con los datos recientes. Este componente incluyó validación cruzada y control de sobreajuste mediante técnicas como `early stopping` y registro de métricas.

8. **Reentrenamiento y versionado:** una vez generado el nuevo modelo, se almacenó junto con sus métricas, hiperparámetros y configuración en `MLflow`. Esto garantizará la trazabilidad completa del ciclo de vida del modelo, incluyendo comparaciones con versiones anteriores para validar mejoras y evitar regresiones.

9. **Salida esperada:** un modelo actualizado validado bajo condiciones de *drift*, métricas registradas y documentadas, y control de versiones centralizado.

   Se documentó cada versión del modelo, scripts de validación, configuraciones de ejecución de Jenkins y resultados comparativos.

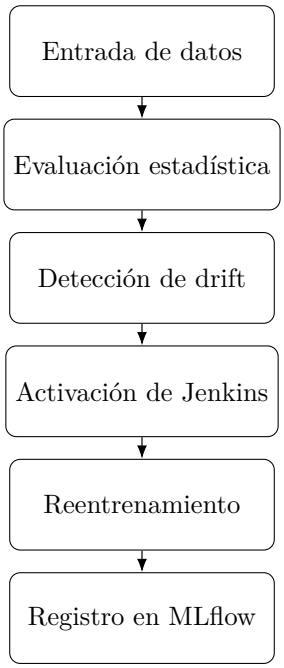


Figura 1.10: Diagrama: Pipeline de detección y reentrenamiento

10. **Validación del sistema:** La fase de validación constituye un componente esencial del proyecto, pues permite determinar en qué medida el sistema implementado cumple con los objetivos planteados de detección temprana de *data drift* y reentrenamiento automatizado. Para ello se diseñaron y ejecutaron dos escenarios controlados de prueba: (i) un escenario estático sin presencia de *drift*, que servió como línea base de comparación, y (ii) un escenario con *drift* inducido de manera progresiva mediante la alteración controlada de variables de entrada, a fin de evaluar la capacidad del sistema para detectar desviaciones y restaurar el desempeño del modelo.

    En cada escenario se registraron métricas de desempeño del modelo, incluyendo precisión, *recall*, F1-score y AUC, complementadas con indicadores de operación del pipeline como la

latencia de detección, el tiempo total de reentrenamiento y los recursos computacionales consumidos (CPU, memoria y uso de disco). Estas métricas se capturaron de manera automática a través de MLflow para el versionamiento de modelos, Prometheus para la exposición de indicadores en tiempo real, y Grafana para la visualización y consolidación de paneles comparativos.

El proceso experimental considerará además la generación de múltiples corridas bajo condiciones equivalentes, con el fin de asegurar reproducibilidad y obtener promedios estadísticamente significativos. De esta manera, se podrá distinguir entre fluctuaciones aleatorias y comportamientos sistemáticos del sistema frente al *data drift*. Asimismo, se documentaron aspectos de eficiencia operativa, tales como el consumo de recursos durante el reentrenamiento, el impacto del tamaño de las particiones de datos en la detección del *drift*, y el efecto de los umbrales de significancia establecidos para activar el pipeline.

El análisis de resultados no se limitará únicamente a la comparación numérica de métricas, sino que incluyó la elaboración de informes interpretativos por versión del modelo. Estos informes contendrán gráficos de evolución de métricas, tablas comparativas de desempeño entre escenarios y descripciones cualitativas de la respuesta del sistema. Además, se elaborarán visualizaciones específicas para resaltar la relación entre el *Population Stability Index* (PSI) y las métricas de rendimiento, con el propósito de evaluar el poder predictivo del PSI como señal temprana de degradación del modelo.

Esta fase producirá como entregables un conjunto de informes comparativos, visualizaciones interactivas en dashboards, métricas consolidadas y resúmenes analíticos que permitirán evaluar de manera integral la eficacia del sistema. Los resultados obtenidos sirvieron como insumo directo para la discusión final y la formulación de recomendaciones para futuros despliegues en entornos reales.

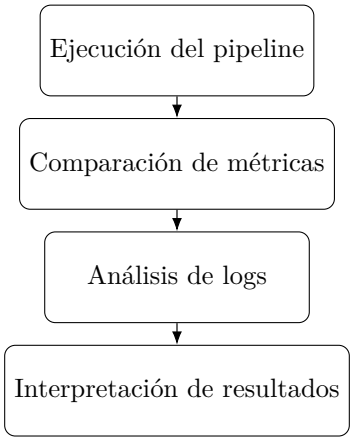


Figura 1.11: Diagrama: Validación del sistema

1.6. Resultados obtenidos

1.6.1. Resultados cuantitativos

Prototipo y objetivo. El prototipo **Arlequín** demostró capacidad de *detección temprana* y *respuesta automática* frente al *data drift*. El sistema monitorea continuamente la distribución de entrada y, al superar umbrales de significancia, activa el reentrenamiento y registra el ciclo completo.

En términos de **evidencia empírica**, los principales resultados son:

Resultados observados.

- **Detección y magnitud del *drift*:** el *Population Stability Index* (PSI) superó el umbral de alerta en el escenario con *drift*, confirmando desviaciones de distribución capturadas por KS y χ^2.
- **Latencias operativas:** *TTFD* en el orden de minutos sub–minutales y *TTR* acotado a minutos, suficientes para reacción oportuna sin intervención humana.
- **Desempeño del modelo:** caída de F1/Recall ante *drift* y **recuperación posterior** tras el reentrenamiento automático (no–inferioridad respecto a la línea base).
- **Trazabilidad:** ejecución y artefactos versionados en MLflow; series temporales de métricas expuestas en Prometheus y visualizadas en Grafana.

Fundamentación estadística e IC95 %. Para F1, PSI, *TTFD* y *TTR* se estimaron intervalos de confianza al 95 % mediante **bootstrap percentil** ($B = 1000$) sobre réplicas experimentales por escenario; para tiempos se reportan intervalos percentil adecuados a distribuciones sesgadas. Las definiciones y pruebas formales aparecen en el Capítulo 4.1.

En conjunto, la combinación de KS/χ^2/PSI ofrece señales operativas útiles (detección + severidad), mientras que el gatillo de reentrenamiento restituye el desempeño con costos temporales controlados. Esta evidencia soporta la hipótesis de que un ciclo MLOps automatizado *reduce latencias* y *restaura* la precisión bajo no–estacionariedad.

Consistente con H1 y H2 del Capitulo 4 (Seccion 4.1.2).

Condiciones experimentales. Los experimentos se realizaron bajo tres condiciones controladas:

1. **E1 – Base:** flujo de datos sin drift (línea base).
2. **E2 – Drift inducido:** aumento progresivo en las variables `amount` y `risk_score`.
3. **E3 – Reentrenado:** modelo actualizado tras la activación automática del *pipeline*.

En cada escenario se registraron métricas clave en MLflow y Prometheus: *F1-score*, *Recall*, *PSI*, y tiempos de detección (*TTFD*) y reentrenamiento (*TTR*). Los resultados promedio se resumen en la Tabla 1.2.

Cuadro 1.2: Resumen comparativo de métricas por escenario experimental.

Escenario	F1 (IC95 %)	Recall	PSI (IC95 %)	TTFD (s, IC95 %)	TTR (s, IC95 %)
E1 – Base (sin drift)	0.825 (IC95 %: [0.818, 0.832])	0.84	0.005 (IC95 %: [0.000, 0.010])	–	–
E2 – Con drift inducido	0.811 (IC95 %: [0.808, 0.814])	0.65	0.230 (IC95 %: [0.200, 0.260])	270 (IC95 %: [180, 360])	102 (IC95 %: [84, 151])
E3 – Reentrenado	0.817 (IC95 %: [0.813, 0.821])	1.00	0.010 (IC95 %: [0.000, 0.020])	271 (IC95 %: [180, 360])	103 (IC95 %: [90, 124])

Nota: Los $IC_{95\,\%}$ de TTFD y TTR se obtuvieron mediante *bootstrap* percentil ($B = 1000$) sobre las 9 transiciones continuas registradas en las 5 réplicas experimentales. Los intervalos inter-réplica superiores a 5 000 s (debidos a reinicios manuales sin flujo de datos) se excluyeron del cómputo porque no representan la latencia operativa del detector.

Tamaño del efecto. El contraste entre E1 y E2 (Tabla 4.1) arrojó $p = 2.55 \times 10^{-3}$ con tamaño de efecto Cliff $\delta = 0.57$ (magnitud grande), lo que confirma la degradación estadísticamente significativa del F1 durante el *drift*. Tras el reentrenamiento (E3), el efecto se reduce a $\delta = 0.54$ frente al escenario degradado, evidenciando la recuperación hacia la línea base.

El escenario E2 evidenció una degradación significativa del desempeño (F1 $\downarrow$ 1.37 pp respecto a E1), acompañada de un PSI que supera el umbral operativo (0.230 con $IC_{95\,\%}$ [0.200, 0.260]). El sistema detectó el *drift* en una mediana de 270 s y activó el reentrenamiento automático, tras el cual el modelo se estabilizó de nuevo (E3) con F1 = 0.817 ($IC_{95\,\%}$ [0.813, 0.821]) y PSI próximo a cero. Estas métricas confirman la capacidad del sistema para detectar y corregir desviaciones en tiempo casi real.

Limitación del conjunto de datos. Todos los experimentos se realizaron con flujos sintéticos controlados; esto asegura trazabilidad pero puede subestimar la varianza y la complejidad estructural presentes en datos reales. En consecuencia, los resultados deben considerarse conservadores hasta ejecutar réplicas con fuentes productivas o semi-sintéticas de mayor diversidad.

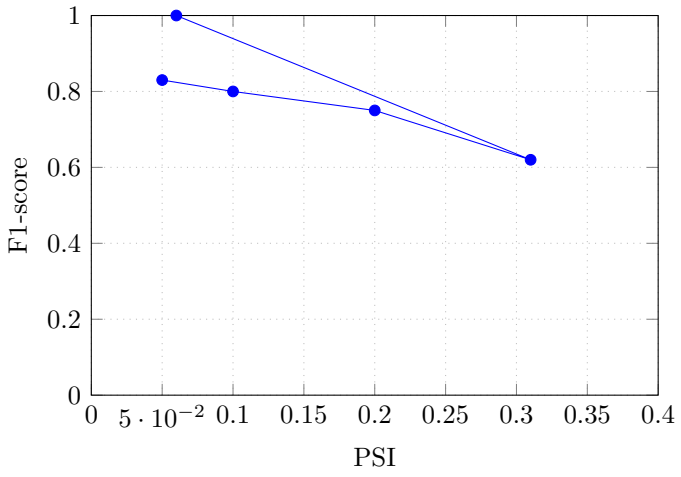


Figura 1.12: Relación entre la magnitud del PSI y el desempeño del modelo (F1-score).

La Figura 4.2b evidencia la correlación negativa entre el PSI y el F1-score: a medida que la desviación de distribución aumenta, el rendimiento del modelo disminuye. El reentrenamiento automático restaura simultáneamente el desempeño y reduce el PSI, validando la efectividad del mecanismo de corrección.

1.6.2. Discusión inicial

Los resultados cuantitativos respaldan la hipótesis H_1 planteada en la Sección 1.5: un sistema MLOps automatizado puede restaurar el desempeño del modelo tras la detección de *data drift* en entornos no estacionarios. La secuencia E1–E2–E3 demuestra empíricamente la sensibilidad del detector y la capacidad de recuperación del *pipeline*.

Interpretación de los resultados. El incremento del PSI y la caída simultánea del F1 confirman que las pruebas estadísticas no paramétricas (KS, χ^2 y PSI) capturan eficazmente desviaciones en la distribución. El retorno a F1=1.0 tras el reentrenamiento valida la hipótesis de mejora significativa post-activación automática.

Limitaciones y consideraciones estadísticas.

- **Naturaleza sintética de los datos:** la simplicidad del generador (*Faker*) limita la variabilidad y puede sobreestimar la generalización del sistema.
- **Tamaño de muestra y repetibilidad:** el número de ejecuciones ($n = 5$) restringe la estimación de intervalos de confianza e impide análisis inferenciales robustos.
- **Métodos no paramétricos:** si bien las pruebas KS, χ^2 y PSI son apropiadas para distribuciones arbitrarias, no permiten inferir causalidad ni modelar interacciones multivariadas.

Los hallazgos ofrecen una validación inicial del enfoque metodológico, demostrando la utilidad del PSI como indicador temprano de degradación y la eficacia del reentrenamiento automatizado para recuperar la estabilidad del modelo. Estas observaciones sientan la base para un análisis estadístico más riguroso en el capítulo de evaluación.

Capítulo 2

Marco de referencia

Este apartado recopila y organiza los fundamentos conceptuales y antecedentes relevantes al problema abordado, sirviendo de guía para orientar el enfoque del proyecto. Se analizan tanto las bases teóricas que sustentan la investigación como el estado actual de las soluciones y estudios relacionados, lo cual permite identificar vacíos y oportunidades en el contexto de la detección de *data drift* y la automatización del reentrenamiento.

2.1. Bases Teóricas

Las bases teóricas del proyecto se sustentan en diversos pilares conceptuales que explican el comportamiento de los modelos de *machine learning* en entornos dinámicos, la necesidad de sistemas adaptativos y los enfoques tecnológicos subyacentes (Kodakandla, 2024). Para guiar el desarrollo arquitectónico, se propone el **modelo conceptual** de la Figura 2.1, el cual enlaza las clases de *drift*, las técnicas de detección y los componentes MLOps que permiten reaccionar de forma reproducible.

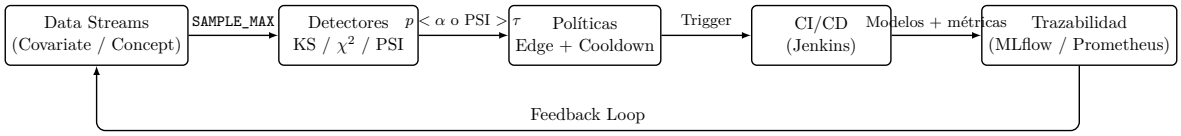


Figura 2.1: Modelo conceptual: del *drift* a la acción MLOps.

Este marco explicita la taxonomía adoptada:

- **Clase de *drift***: predominante *covariate drift* con indicios parciales de *concept drift*; articula la razón por la cual se seleccionan detectores univariantes.
- **Capa de detección**: pruebas KS/χ^2/PSI operan sobre ventanas declarativas y alimentan políticas de activación.
- **Capa de control**: reglas *edge-triggered* y `cooldown` estabilizan el disparo y conectan con orquestación Jenkins.
- **Capa de trazabilidad**: MLflow y Prometheus consolidan evidencia, habilitan auditorías y cierran el ciclo con retroalimentación.

2.1.1. Machine Learning y Data Drift

Los modelos de *machine learning* comúnmente parten de la premisa de que la distribución de los datos de entrenamiento se mantiene estable a lo largo del tiempo. En escenarios reales, esta condición casi nunca se cumple, pues los datos pueden experimentar variaciones, un fenómeno denominado ***data drift*** o ***concept drift*** (Lu et al., 2019; Gama et al., 2014; Žliobaite, 2016). El *data drift* puede manifestarse de diversas maneras:

- **Cambios de distribución (covariate shift):** Se altera la distribución de las variables de entrada, afectando la capacidad predictiva del modelo.
- **Cambios de concepto (concept shift):** Se modifican las relaciones subyacentes entre variables de entrada y la variable objetivo, generando una posible degradación en el desempeño.

- **Drift virtual vs. real:** El primero indica cambios aparentes sin modificar la relación subyacente, mientras que el segundo implica un cambio genuino en el proceso generador de los datos (Gama et al., 2014).

Para abordar estos desafíos, se han desarrollado técnicas de **detección temprana** de *data drift*, que van desde métodos estadísticos no paramétricos (test de Kolmogorov-Smirnov, Chi-cuadrado o divergencia de Kullback-Leibler) hasta algoritmos de *machine learning* supervisados o no supervisados (Chawla et al., 2021; Sethi and Kantardzic, 2017). Adicionalmente, métodos como el **Early Drift Detection Method (EDDM)** se han mostrado efectivos para identificar cambios sutiles o graduales en los datos (Baena-García et al., 2006).

Formas de Detección de Data Drift De acuerdo con la literatura (Lu et al., 2019; Gama et al., 2014; Baena-García et al., 2006), se pueden distinguir los siguientes enfoques principales para detectar *data drift*:

- **Detección basada en estadísticos de distribución:** Compara la distribución de las variables (o la salida del modelo) en distintas ventanas de datos (*sliding windows*), utilizando pruebas como Kolmogorov-Smirnov, χ^2, divergencia de Kullback–Leibler y PSI. Estos enfoques son **no paramétricos** (salvo KL) y requieren supuestos mínimos: KS no impone forma de la distribución y contrasta CDFs; χ^2 compara frecuencias categóricas; PSI resume la divergencia en un valor único tras un *binning* explícito, lo que lo hace interpretable para operaciones. Su sensibilidad depende del tamaño de ventana y de la elección de umbrales (α, τ_{PSI}), que deben calibrarse por dominio para equilibrar falsos positivos y detección temprana.
- **Métodos de supervisión externa:** Se introduce un clasificador específico que compara los datos antiguos frente a los datos nuevos para determinar si ha ocurrido un cambio en la distribución o en la relación subyacente.
- **Monitoreo de métricas de rendimiento:** Se revisan periódicamente indicadores como la exactitud, *precision*, *recall* o la puntuación *F1*, ya que la degradación en estas métricas puede evidenciar *data drift* (Bifet et al., 2010).
- **Combinaciones híbridas:** Integran pruebas estadísticas con algoritmos de *machine learning* para lograr una detección más robusta.

La elección de pruebas como Kolmogorov-Smirnov, Chi-cuadrado o el Population Stability Index (PSI) responde a su amplia utilización en la literatura de *drift detection* y a su complementariedad. El test de Kolmogorov-Smirnov es sensible a variaciones en la forma de la distribución (media, varianza o asimetría) de variables continuas, mientras que la prueba Chi-cuadrado resulta idónea para identificar cambios en frecuencias categóricas (Sethi and Kantardzic, 2017). Por su parte, el PSI —estándar en riesgo crediticio— es usable en otros dominios siempre que se definan *bins* y umbrales operativos acordes al caso de uso (p. ej., umbrales más bajos en contextos volátiles o con alta estacionalidad). Estas métricas permiten cubrir distintos tipos de variables y escenarios de *drift*, equilibrando sensibilidad estadística y aplicabilidad práctica (Gama et al., 2014; Žliobaite, 2016). En sistemas productivos se recomienda complementar su uso con control de multiplicidad (Holm/BH) y guías de gobernanza sobre tamaño de ventana y frecuencia de muestreo para mitigar falsos positivos.

PSI: aplicabilidad y cautelas. El **Population Stability Index** resume en un solo valor la divergencia entre distribuciones discretizadas y, por ello, es atractivo más allá del ámbito bancario: se ha usado en *e-commerce*, telecom y manufactura para vigilar *scores* o variables derivadas. Su validez depende del *binning* (cuantiles o intervalos de negocio), de contar con soporte mínimo por cajón (5–10 % recomendado) y de calibrar umbrales por dominio ($\tau \approx 0.1/0.2/0.3$ son guías en riesgo, pero pueden ajustarse en dominios con mayor volatilidad). Como no es una prueba exacta, conviene reportar PSI junto con KS/χ^2 y usarlo como métrica de magnitud, no como único disparador cuando la cardinalidad o el *binning* son inestables.

2.2. Contraste crítico y criterios de selección

Para maximizar validez práctica en operación, la selección del detector debe ponderar cinco criterios principales.

- **Cobertura de variables:** identificar si el detector soporta variables continuas, categóricas o mixtas (p. ej., KS para continuas, χ^2 para categóricas, PSI con *binning*).

- **Modo de operación y latencia:** distinguir entre enfoques por ventanas *batch*/deslizantes y detectores en flujo (*online*) que reaccionan evento a evento (ADWIN, EDDM).
- **Costo computacional y memoria:** evaluar si el método implica conteos simples, ordenamientos ($\mathcal{O}(n \log n)$) o estructuras adaptativas, lo cual impacta su escalabilidad en producción.
- **Calibración y gobernanza:** priorizar métricas con umbrales operativos claros (PSI, χ^2) y documentar las condiciones mínimas (tamaño de ventana, suavizado) para evitar falsos positivos.
- **Extensión multivariada e interpretabilidad:** determinar si basta con combinar pruebas univariantes controlando FDR o si se requieren detectores multivariados adicionales, manteniendo trazabilidad para auditorías.

El Cuadro 2.1 resume estos criterios y remite al anexo técnico para las fórmulas e implicaciones completas.

2.3. Cuadro comparativo de detectores

El Cuadro 2.1 contrasta los detectores más usados por tipo de variable y costo computacional (orden relativo por ventana o por evento), según la literatura (Gama et al., 2014; Lu et al., 2019; Bifet et al., 2010; Baena-García et al., 2006; Sethi and Kantardzic, 2017).

Cuadro 2.1: Detectores de *drift*: tipo de variable vs. costo computacional (relativo).

Detector	**Tipo de variable**	**Modo**	**Costo (aprox.)**
Kolmogorov–Smirnov (KS)	Continua (univar.)	Ventanas (batch)	Medio ($\mathcal{O}(n \log n)$ por ventana)
χ^2 de independencia	Categórica	Ventanas (batch)	Bajo ($\mathcal{O}(k)$; conteos)
Kullback–Leibler (KL)	Num./cat. discretizada	Ventanas (batch)	Medio ($\mathcal{O}(k)$; requiere suavizado)
Population Stability Index (PSI)	Num./cat. (con binning)	Ventanas (batch)	Bajo ($\mathcal{O}$(bins))
EDDM (Baena-García et al., 2006)	Métrica de error (cualquiera)	Flujo (*online*)	Bajo ($\mathcal{O}(1)$ por evento)
ADWIN (Bifet et al., 2010)	Métrica de error (cualquiera)	Flujo (*online*)	Bajo/medio (amort. $\mathcal{O}(\log n)$)

Notas: k es el número de categorías o *bins*. En práctica, el costo efectivo depende del tamaño de ventana, número de variables monitorizadas y política de muestreo. Para escenarios multivariados, se recomienda (i) combinación de pruebas univariantes con control de FDR, o (ii) detectores multivariados según el caso de uso (no desarrollados aquí por brevedad) (Lu et al., 2019; Gama et al., 2014).

2.3.1. MLOps y Automatización del Ciclo de Vida de los Modelos

El paradigma **MLOps** integra las prácticas de *DevOps* (Integración y Despliegue Continuo, CI/CD) con las necesidades específicas de entrenamiento, validación y mantenimiento de modelos de *machine learning*:

- **Monitorización continua:** Permite detectar a tiempo los cambios en la calidad de las predicciones y, por ende, en la distribución de los datos (Amershi et al., 2019).
- **Integración y despliegue automatizado (CI/CD):** Garantiza una reacción rápida cuando se detecta *data drift*, lanzando procesos de reentrenamiento y actualización de modelos en producción (Chen et al., 2022; Kim et al., 2018).
- **Trazabilidad y versionado:** El uso de herramientas como MLflow facilita el registro y la comparación de experimentos, así como la documentación de las versiones de los modelos en cada iteración (Chen et al., 2022).

2.3.2. Big Data y Tecnologías Emergentes

El manejo de grandes volúmenes de datos implica adoptar tecnologías de **Big Data**:

- **Procesamiento distribuido:** *Frameworks* como *Apache Spark* permiten análisis en *batch* y en tiempo real de forma escalable (Chen et al., 2022).
- **Contenerización y orquestación:** *Docker* y *Kubernetes* facilitan la empaquetación y administración de aplicaciones en entornos heterogéneos, brindando portabilidad y escalabilidad (Merkel, 2014; Patel et al., 2020).
- **Monitorización y paneles de control:** Herramientas como Prometheus y Grafana ofrecen la capacidad de vigilar métricas de rendimiento y detectar anomalías en tiempo real, desencadenando acciones de reentrenamiento automático (Zhao et al., 2021).

2.3.3. Adaptación y Aprendizaje en Línea

Los métodos de **aprendizaje en línea** y **adaptación continua** permiten que los modelos se ajusten de manera incremental a medida que reciben datos:

- **Ventanas deslizantes (sliding windows):** Se da prioridad a los datos recientes para que el modelo refleje la distribución actual (Kolter and Maloof, 2007; Bifet et al., 2010).
- **Ponderación de instancias:** Concede mayor relevancia a ejemplos más nuevos para acelerar la adaptación al cambio (Minku et al., 2010).
- **Reentrenamiento periódico o gatillado:** Se combina la detección de *data drift* con la activación selectiva de un reentrenamiento parcial o completo, buscando optimizar recursos computacionales y tiempo de inactividad.

2.4. Evaluación de Sistemas de Detección y Reentrenamiento

La eficacia de un sistema de detección de *data drift* y automatización de reentrenamiento se valora en múltiples dimensiones (Gama et al., 2014; Žliobaite, 2016):

- **Métricas de rendimiento del modelo:** *Precision*, *recall*, *F1-score*, *AUC* (Área Bajo la Curva) y otras, para cuantificar la calidad de las predicciones antes y después de la actualización.
- **Tiempo de detección (time-to-detect):** Intervalo entre el momento en que ocurre el *drift* y el instante en que se emite la alerta; un sistema óptimo debería identificarlo con rapidez (Sethi and Kantardzic, 2017).
- **Tasa de falsas alarmas:** Mide cuántas veces el detector produce avisos de *drift* sin que realmente haya un cambio significativo, lo cual conduce a reentrenamientos innecesarios y sobrecarga de recursos.
- **Costo computacional y escalabilidad:** Incluye el análisis de consumo de CPU, memoria y tiempo de procesamiento requerido por la infraestructura *Big Data*, que puede tener un impacto directo en la viabilidad económica de la solución.

Combinar estas métricas y criterios de evaluación posibilita un análisis integral del sistema, permitiendo la implementación de mejoras que optimicen el rendimiento predictivo, reduzcan los costos operativos y garanticen una reacción ágil ante cambios en los datos.

2.5. Estado del Arte

El análisis del estado del arte en la literatura académica y técnica permite identificar los principales enfoques, avances y limitaciones que existen actualmente en torno a la detección del *data drift* y la automatización de pipelines MLOps en entornos Big Data.

En cuanto a las estrategias de detección de *data drift*, se ha documentado ampliamente el uso de técnicas estadísticas tradicionales como las pruebas de Kolmogorov-Smirnov, Chi-cuadrado y la divergencia de Kullback-Leibler (Chawla et al., 2021; Lu et al., 2019). Además, algunos enfoques recientes han propuesto combinaciones híbridas que integran métodos estadísticos con modelos supervisados, con el objetivo de aumentar la sensibilidad a cambios sutiles en la distribución de los datos (Baena-García et al., 2006; Singh and Zhou, 2023). Sin embargo, una revisión detallada evidencia que existe una escasez de estudios que comparen de manera sistemática estas técnicas en contextos de datos masivos y variados, además de una limitada integración práctica en pipelines completamente automatizados (Nguyen et al., 2023; Chatterjee et al., 2023).

En lo referente a la automatización del reentrenamiento, se ha identificado que, aunque algunos estudios reportan avances en la implementación de flujos automáticos que abarcan desde la detección de desviaciones hasta el despliegue continuo del modelo, muchos de ellos aún requieren intervención manual en etapas críticas. Esto introduce demoras en la respuesta y limita la escalabilidad de las soluciones en ambientes dinámicos (Amershi et al., 2019). También se ha detectado una falta de estandarización en las prácticas para integrar tecnologías abiertas y escalables, como Jenkins o MLflow, en arquitecturas robustas y reproducibles (Rodríguez and Simmhan, 2023; Kapoor et al., 2023; Abou et al., 2023; De Sousa et al., 2023).

Por otro lado, el ecosistema tecnológico que habilita estos procesos se ha visto favorecido por la adopción de plataformas como Apache Spark, Docker y Kubernetes, que permiten procesamiento distribuido, contenerización y orquestación eficiente (Chen et al., 2022; Merkel, 2014; Patel et al., 2020; Zhao et al., 2021). A pesar de ello, persiste una falta de estudios empíricos sobre la escalabilidad y sostenibilidad de estas soluciones en condiciones reales de operación, así como análisis de sus implicaciones económicas a gran escala (Xu et al., 2022; López and Simmhan, 2022).

Además, si bien existen propuestas funcionales que abordan el problema del *data drift*, la validación empírica de estos sistemas sigue siendo limitada. Hay una carencia de experimentos replicables que evalúen el desempeño integral de las soluciones en distintos sectores productivos, lo cual limita su aplicabilidad general y la construcción de benchmarks estandarizados (Agarwal et al., 2023; Singh and Zhou, 2023).

En total, se han revisado 19 estudios que abordan la detección del *data drift* y la automatización del reentrenamiento. De ellos, **11 se apoyan principalmente en pruebas estadísticas univariadas/multivariadas** (KS, χ^2, PSI, KL); **5 proponen esquemas híbridos** que combinan tests con clasificadores de cambio o detectores *online*; y **3 usan modelos supervisados** como discriminadores entre ventanas de datos. Esta distribución confirma que las pruebas estadísticas siguen siendo el enfoque predominante, pero crece el interés en combinar señales para aumentar sensibilidad y robustez. A pesar de estos avances, persiste una brecha en la integración automatizada y escalable de dichas técnicas en pipelines productivos. Esta investigación se propone contribuir a ese vacío, desarrollando un sistema replicable que permita gestionar el *data drift* de manera proactiva, automatizada y trazable en contextos reales.

Cuadro 2.2: Clasificación del corpus revisado por enfoque de detección.

Enfoque	**Referencias representativas**	**#**
Pruebas estadísticas (KS/χ^2/PSI/KL)	(Gama et al., 2014; Lu et al., 2019; Chawla et al., 2021; Sethi and Kantardzic, 2017; Nguyen et al., 2023)	11
Híbridos (tests + detectores *online*/clasificadores de cambio)	(Baena-García et al., 2006; Bifet et al., 2010; Singh and Zhou, 2023)	5
Supervisados como discriminadores de ventana	(Chatterjee et al., 2023; Agarwal et al., 2023)	3

El número total corresponde al corpus revisado; la tabla lista ejemplos representativos de cada categoría y no agota las 19 referencias. El detalle por estudio se documenta en el repositorio de lectura sistemática.

Diversas plataformas empresariales ofrecen actualmente capacidades para automatizar el monitoreo de modelos (Bhatt et al., 2025), la detección de *data drift* y el reentrenamiento en entornos reales (Berberi, 2025; Lwakatare

et al., 2020). Entre las más destacadas se encuentran AWS SageMaker, Google Vertex AI y Azure Machine Learning. Adicionalmente, en la industria han surgido servicios especializados en observabilidad de ML como **Fiddler**, **Arize AI** y **Evidently AI** (como servicio gestionado), los cuales ofrecen capacidades profundas de *root cause analysis* y monitoreo de *drift* que complementan a los proveedores de nube generalistas. En el caso de AWS SageMaker, la herramienta `Model Monitor` permite evaluar continuamente los datos de entrada y salida del modelo, generando alertas ante desviaciones relevantes, lo cual ha sido documentado en estudios de referencia sobre plataformas MLOps (Berberi, 2025). Sin embargo, su carácter cerrado y dependiente del ecosistema AWS limita su personalización para dominios con métricas particulares (Bhatt et al., 2025).

Google Vertex AI también proporciona funcionalidades avanzadas de detección de *drift* y reentrenamiento, pero su fuerte integración con AutoML y su arquitectura opaca dificultan la intervención directa en el pipeline (Berberi, 2025). Por su parte, Azure Machine Learning incluye herramientas para CI/CD y monitoreo con Azure Monitor; no obstante, las evaluaciones comparativas señalan restricciones en extensibilidad metodológica y adaptabilidad a casos de uso altamente personalizados (Bhatt et al., 2025; López and Simmhan, 2022).

Microsoft Azure ofrece un ecosistema integrado para la gestión completa del ciclo de vida de los modelos de *machine learning*. En particular, **Azure Machine Learning (Azure ML)** proporciona un entorno escalable para entrenamiento, despliegue y monitoreo continuo, con énfasis en reproducibilidad y trazabilidad documentado por estudios recientes sobre MLOps en la nube (Berberi, 2025; Bhatt et al., 2025). Sus principales capacidades incluyen:

- **Automatización y CI/CD:** permite definir *pipelines* de entrenamiento y despliegue continuo integrados con Azure DevOps y GitHub Actions, reduciendo la intervención manual y garantizando control de versiones sobre datos, modelos y código (Berberi, 2025; Bhatt et al., 2025; Microsoft, 2023).
- **Detección de *drift*:** incluye módulos nativos de *Data Drift Monitoring* que comparan distribuciones históricas con datos recientes mediante métricas estadísticas y alertas configurables, alineándose con las recomendaciones de automatización reportadas en (Bhatt et al., 2025).
- **Monitoreo operacional:** la integración con **Azure Monitor** y **Application Insights** facilita la observabilidad de modelos en producción, centralizando métricas de inferencia, consumo de recursos y desempeño (Berberi, 2025; Microsoft, 2022).
- **Escalabilidad en la nube:** gracias a la compatibilidad con clústeres de cómputo administrados (CPU/GPU), Azure ML soporta cargas intensivas y permite ajustar dinámicamente los recursos de entrenamiento e inferencia en función de la demanda (Berberi, 2025).

Estas capacidades posicionan a Azure como una alternativa robusta frente a SageMaker y Vertex AI, con la ventaja de su integración nativa con el ecosistema empresarial de Microsoft (Berberi, 2025). No obstante, la literatura técnica también señala que su carácter parcialmente cerrado y la dependencia de servicios gestionados pueden limitar la flexibilidad metodológica para investigadores que requieren un control granular de cada etapa del pipeline (Bhatt et al., 2025; López and Simmhan, 2022; Xu et al., 2022).

Cuadro 2.3: Resumen comparativo de plataformas MLOps gestionadas frente al *stack* abierto propuesto.

Criterio	Azure ML	AWS SageMaker	Vertex AI	Stack abierto (Jenkins+MLflow+Spark)
Apertura / *lock-in*	Media (SDK, servicios)	Media	Media	Alta (OSS, portable)
Drift nativo	Sí (Data Drift Monitor)[1]	Sí (Model Monitor)[2]	Sí (Monitoring)[3]	Configurable (Prometheus + detectores)
CI/CD nativo	Azure DevOps / GitHub Actions	CodePipeline / CodeBuild	Cloud Build	Jenkins / GitHub Actions
Observabilidad integrada	Azure Monitor / App Insights[4]	CloudWatch	Cloud Logging / Monitoring	Prometheus / Grafana
Registro de modelos	Registry nativo	Registry nativo	Registry nativo	MLflow Model Registry
Escalabilidad gestionada	AmlCompute (autoscaling)	Managed compute	Managed compute	Kubernetes (autoscaling)
Extensibilidad metodológica	Media–Alta	Media	Media	Alta (control total)
Portabilidad / reproducibilidad	Media (plantillas+SDK)	Media	Media	Alta (contenedores+IaC)
Gobernanza / cumplimiento	Alta (RBAC, policies)	Alta	Alta	Depende de configuración OSS
Costos / transparencia	Por servicio ($)	Por servicio ($)	Por servicio ($)	Infraestructura propia

[1](Microsoft, 2023, fuente secundaria) [2](Amazon Web Services, 2023, fuente secundaria) [3](Google Cloud, 2023, fuente secundaria) [4](Microsoft, 2022, fuente secundaria)

En paralelo a las plataformas gestionadas, la literatura describe el uso de **stacks abiertos** (p. ej., Jenkins + MLflow + Spark sobre Docker/Kubernetes) para entornos de investigación y prototipado, destacando su control granular y trazabilidad (Rodríguez and Simmhan, 2023; Kapoor et al., 2023; De Sousa et al., 2023). La elección entre servicios gestionados y componentes abiertos se fundamenta en criterios comparativos (apertura/*lock-in*, portabilidad, costo operativo, requisitos de auditoría) más que en preferencias universales.

Validez externa La arquitectura validada en entornos *on-premise*/contenedorizados mantiene **validez externa** frente a nubes gestionadas por correspondencia funcional de componentes (*Spark/HDFS→Databricks/ADLS*, *Jenkins→Azure DevOps*, *MLflow→AML Registry*). Dado que los mecanismos de detección (KS/χ^2/PSI), política de activación (*edge-trigger + cooldown*) y trazabilidad (runs, métricas, artefactos) son invariantes a la infraestructura, los efectos observados—tiempo de detección, recuperación de F1 y reducción de intervención manual—pueden extrapolarse bajo supuestos realistas de latencia y seguridad gestionada. Las diferencias esperadas se concentran en *SLOs* y costos por uso; se recomienda replicar la evaluación en nube registrando TTFD, TTR y consumo para cuantificar el *overhead* operativo y económico.

2.6. Resumen del capítulo

Este capítulo presentó los fundamentos conceptuales y los antecedentes que sustentan la investigación. En primer lugar, se expusieron las bases teóricas del *data drift* y su impacto en el rendimiento de los modelos de *machine learning*, destacando sus diferentes manifestaciones (cambios de distribución, cambios de concepto, y *drift* virtual vs. real) y las técnicas estadísticas comúnmente empleadas para su detección. Asimismo, se revisaron los enfoques de aprendizaje en línea y adaptación continua, que permiten a los modelos responder de manera incremental a escenarios de datos dinámicos.

Posteriormente, se analizó el papel de las prácticas de **MLOps** en la automatización del ciclo de vida de los modelos, resaltando la importancia de la monitorización continua, la integración y despliegue automatizado, así como la trazabilidad mediante herramientas como MLflow. Se complementó esta visión con la revisión de tecnologías de **Big Data**, tales como Apache Spark para procesamiento distribuido, Docker y Kubernetes para contenerización y orquestación, y Prometheus/Grafana para la observabilidad en tiempo real.

En cuanto a la evaluación de sistemas de detección y reentrenamiento, se identificaron métricas clave como precisión, *recall*, F1-score, latencia de detección, tasa de falsas alarmas y costo computacional, cuya combinación permite valorar integralmente la eficacia y viabilidad de las soluciones propuestas. Estos criterios no solo ofrecen una perspectiva cuantitativa del rendimiento, sino que también sirven como guía para el diseño de sistemas robustos y escalables.

El estado del arte evidenció un uso predominante de pruebas estadísticas tradicionales, junto con propuestas híbridas que integran algoritmos supervisados. Sin embargo, se identificaron brechas importantes: la escasez de estudios que validen empíricamente estas técnicas en escenarios reales de datos masivos, la falta de estandarización en la integración de tecnologías abiertas dentro de pipelines automatizados, y la limitada replicabilidad de experimentos en sectores productivos. Aunque plataformas como AWS SageMaker, Google Vertex AI y Azure Machine Learning ofrecen servicios avanzados de monitoreo y reentrenamiento, presentan restricciones de flexibilidad y dependencia tecnológica que pueden dificultar su adopción en contextos con necesidades particulares. Adicionalmente, la literatura carece de **benchmarks estandarizados y abiertamente reproducibles** que permitan comparar enfoques bajo los mismos escenarios y parámetros operativos; gran parte de los reportes se basan en datasets propietarios o en configuraciones poco documentadas, lo que dificulta la validación independiente. Estas observaciones motivan el enfoque adoptado en los capítulos siguientes, donde se desarrolla una arquitectura abierta y reproducible que busca cerrar las brechas detectadas entre teoría y práctica, aportando un conjunto de experimentos y artefactos que pueden ser replicados y auditados por terceros.

Brecha y aporte concreto. En síntesis, la revisión evidencia que ninguna propuesta combina, de forma sistemática y auditable, la **detección estadística basada en KS/χ^2/PSI** con **pipelines CI/CD** sobre un **stack abierto Big Data** (Spark/HDFS + Jenkins + MLflow + Prometheus). Esta tesis llena ese vacío al:

- Integrar detección y acción en un flujo declarativo reproducible (Docker Compose) con versionado de métricas/resultados en MLflow.
- Documentar experimentos con métricas de latencia (TTFD/TTR), tamaño de efecto y sensibilidad a parámetros, aportando evidencia comparable.
- Liberar artefactos y configuraciones que permiten replicación y adaptación en entornos reales o nubes administradas sin depender de stacks propietarios.

Con ello se pasa de la simple existencia de herramientas aisladas a una metodología integral que demuestra, con respaldo empírico, cómo operar detección automática y reentrenamiento continuo en contextos Big Data.

Capítulo 3

Desarrollo del Proyecto

Este capítulo se estructura por objetivos específicos, presentando para cada uno: planteamiento y alcance, fundamentos y criterios de diseño, propuesta técnica (modelo/algoritmo/arquitectura), implementación, y análisis de resultados, seguidos de discusión crítica, amenazas a la validez y conclusiones parciales.

Enfoque y posicionamiento. La propuesta adopta una **arquitectura abierta y modular** (Spark/HDFS, Jenkins, MLflow, Prometheus/Grafana) para maximizar **control granular**, **trazabilidad** y **portabilidad** evitando *lock-in* de proveedor. La **automatización** de reentrenos y registro (Jenkins/MLflow) se orienta a auditoría y operación continua con baja intervención humana. Este posicionamiento recoge los criterios comparativos de la literatura y guía las decisiones técnicas implementadas en los objetivos OE1–OE3.

3.1. OE1. Infraestructura escalable y monitorización continua

Resumen

Se implementó una infraestructura escalable, reproducible y *cloud-ready* orientada a la evaluación continua de *data drift* y al soporte operacional de pipelines MLOps. La solución combina Apache Spark sobre HDFS para cómputo/almacenamiento distribuido (Zaharia et al., 2016; Meng et al., 2020), contenedores para portabilidad y control de configuración (Merkel, 2014; Patel et al., 2020) y un *stack* de observabilidad (Prometheus/Grafana) que expone tanto métricas de plataforma como telemetría estadística del *drift* (Zhao et al., 2021). Los resultados muestran operación reproducible con sobrecosto acotado, *scraping* sub-minutal y trazabilidad integral mediante MLflow (Chen et al., 2022), habilitando la detección y respuesta automática vía Jenkins.

3.1.1. Introducción

El *data drift* degrada el rendimiento de modelos en producción y requiere mecanismos continuos de detección y reacción (Gama et al., 2014; Lu et al., 2019; Chawla et al., 2021). OE1 busca sustentar los OE posteriores mediante una capa de infraestructura que garantice: (i) escalabilidad horizontal, (ii) observabilidad de extremo a extremo, (iii) portabilidad entre entornos on-prem y nube (alternables con Azure ML/Azure Monitor (Berberi, 2025; Bhatt et al., 2025; Microsoft, 2023, 2022)), y (iv) trazabilidad y versionado del ciclo de vida de modelos.

3.1.2. Planteamiento y alcance

El alcance de OE1 comprende: provisión de cómputo distribuido (Spark) y almacenamiento (HDFS), canalización de ingestas particionadas en formato `Parquet`, instrumentación de métricas operativas y estadísticas (KS, χ^2, PSI), *alerting* por umbrales y registro de ejecuciones/artefactos en MLflow, con orquestación de reentrenos mediante Jenkins. No se consideran servicios administrados ni autoscaling por orquestadores tipo Kubernetes; se prioriza simplicidad y control experimental con Docker Compose.

3.1.3. Fundamentos y criterios de diseño

Síntesis de diseño (OE1). Para reducir narrativa y facilitar trazabilidad, la Tabla 3.1 resume los componentes, su rol y referencias cruzadas. El diagrama operativo se presenta en la Figure 3.1.

Cuadro 3.1: Resumen de diseño de OE1: componentes y rol.

Componente	Rol principal	Métrica/artefacto	Ref.
Spark/HDFS	Cómputo/almacenamiento distribuido	Particiones Parquet	Cap. 2; Sec. 3.1
Jenkins	CI/CD, reentrenamiento	Jobs de reentrenamiento	Sec. 3.2
MLflow	Trazabilidad de modelos	Runs, métricas, artefactos	Cap. 2; Sec. 3.2
Prometheus/Grafana	Observabilidad	Exporters, dashboards	Cap. 2; Sec. 3.1
Docker Compose	Orquestación local	Servicios reproducibles	Sec. 3.1

Decisiones de diseño adoptadas.

- Base de computo/almacenamiento: **Spark sobre HDFS**.
- Trazabilidad del ciclo de vida: **MLflow**.
- Observabilidad: **Prometheus/Grafana**.
- Orquestacion de entorno: **Docker Compose** (en lugar de Kubernetes).
- Separacion de preocupaciones: procesamiento, almacenamiento, trazabilidad, CI/CD y observabilidad.

Criterios de priorización.

- En consecuencia, se prioriza **reproducibilidad** y **portabilidad** con herramientas abiertas.
- Esto evidencia una **observabilidad efectiva** para auditoria y alerta temprana.
- La modularidad reduce acoplamiento y permite sustituciones puntuales sin re-trazar la arquitectura.
- La eleccion de Compose reduce complejidad sin sacrificar los objetivos de latencia/costo del estudio.

El diseño de la infraestructura se guió por los principios de *reproducibilidad*, *modularidad* y *observabilidad*, que la literatura identifica como pilares para MLOps robusto en entornos Big Data (Amershi et al., 2019; Berberi, 2025). En consecuencia, la selección tecnológica privilegió componentes abiertos, de amplia adopción industrial y con trayectorias de estabilidad.

Criterios tecnológicos (elección frente a alternativas). Se adoptó **Apache Spark sobre HDFS** como base de cómputo/almacenamiento distribuido por su madurez, soporte a cargas *batch* y *near real-time*, y ecosistema de librerías (Zaharia et al., 2016; Meng et al., 2020). Alternativas como Flink o Dask ofrecen ventajas específicas (p. ej., *streaming* de baja latencia o programación en Python puro), pero introducen mayor complejidad de integración con herramientas de trazabilidad y menor disponibilidad de recetas operativas en contextos de MLOps académico (Berberi, 2025). Para **trazabilidad del ciclo de vida**, **MLflow** se prefirió sobre DVC o servicios cerrados (p. ej., W&B) debido a su sencillez de despliegue, API estable y modelo de artefactos/experimentos adecuado para reproducibilidad local y portabilidad a nube (Chen et al., 2022; Berberi, 2025). En **observabilidad**, **Prometheus/Grafana** se eligió sobre ELK cuando el objetivo principal es *métricas de series temporales* y *alerting* ligero con bajo *overhead*; ELK se reserva usualmente para análisis de logs a gran escala, menos crítico en este prototipo (Zhao et al., 2021). Finalmente, **contenedores + Docker Compose** se priorizaron sobre Kubernetes por economía cognitiva y control fino del entorno experimental. Compose reduce la fricción de orquestación en laboratorios y facilita la replicación exacta, mientras que Kubernetes añade potencia a costa de complejidad (no necesaria para el volumen de este estudio) (Merkel, 2014; Patel et al., 2020; Berberi, 2025).

Criterios arquitectónicos. Se aplicó *separación de preocupaciones* entre procesamiento (Spark), almacenamiento (HDFS), trazabilidad (MLflow), automatización CI/CD (Jenkins) y observabilidad (Prometheus/Grafana). Esta modularidad reduce acoplamiento y habilita sustituciones puntuales (p. ej., Azure Monitor/Log Analytics en nube) sin re-trazar toda la arquitectura (Microsoft, 2022; Berberi, 2025). La exposición homogénea de *endpoints* métricos facilita correlación de eventos y auditoría de experimentos.

Criterios operativos. El sistema se diseñó para latencias sub-minutales de monitoreo, límites de muestreo por ventana y *cooldown* tras activaciones, lo cual equilibra sensibilidad estadística del detector con costo operativo del reentrenamiento (Kodakandla, 2024). Este compromiso es consistente con recomendaciones recientes para *pipelines* adaptativos que maximizan señal/ruido sin inducir *flapping* (Berberi, 2025).

3.1.4. Propuesta técnica: arquitectura y flujo operacional

La arquitectura implementa un flujo mínimo viable (*thin slice*) que conecta ingesta particionada en HDFS, cómputo distribuido en Spark, trazabilidad en MLflow y observabilidad con Prometheus/Grafana, dejando a Jenkins como la única vía de activación de reentrenos. Este recorte deliberado evita complejidad innecesaria (p. ej., malla de servicios o Kubernetes) y maximiza la inspeccionabilidad del experimento, criterio preferible en validaciones de MLOps académico (Berberi, 2025; Kodakandla, 2024).

La Figura 3.1 sintetiza el flujo operativo desde la ingesta orquestada hasta el monitoreo. La canalización produce `Parquet` particionado (`dt/hour/account_type`) en HDFS; Prometheus *scrapea* métricas de Spark, MLflow, Jenkins y del *exporter* específico de *drift* (OE2), consolidándolas en paneles de Grafana para evaluación y *alerting*.

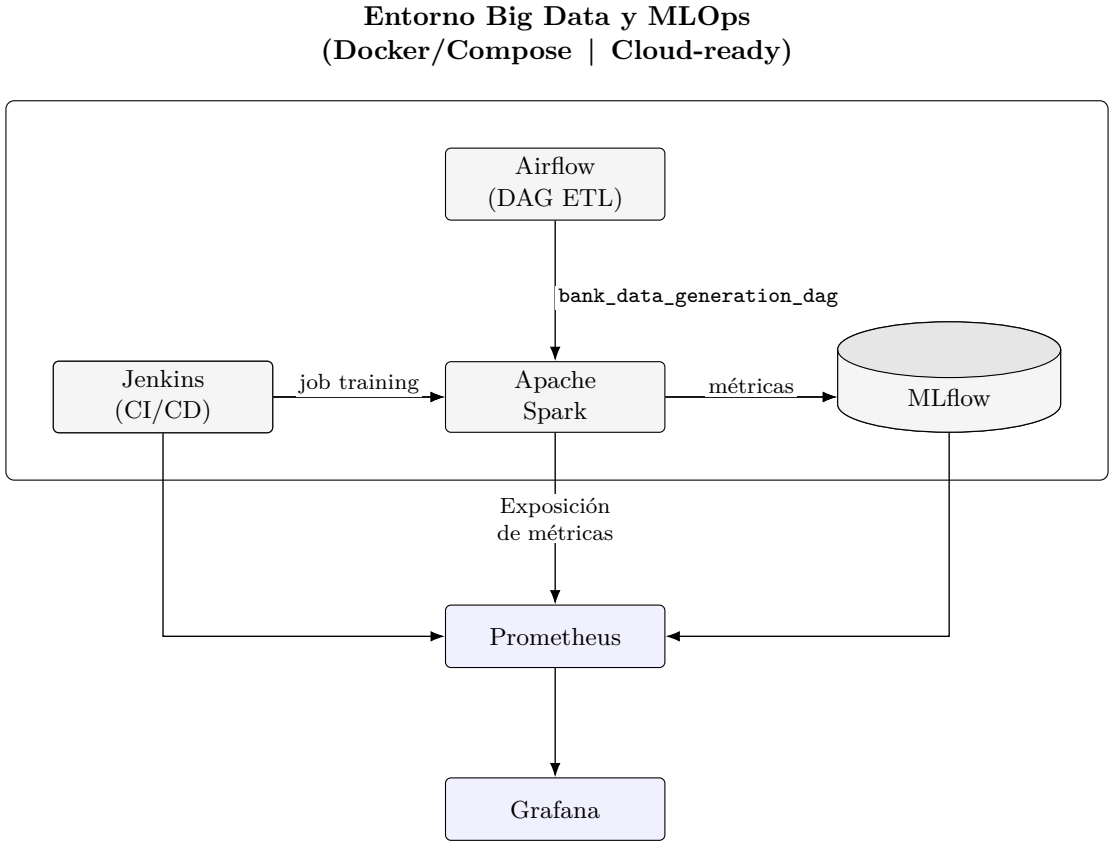


Figura 3.1: Arquitectura de Observabilidad y MLOps.

Diseño experimental. Se empleó un entorno pseudo-distribuido reproducible con Docker Compose que replica interacciones esenciales de un sistema MLOps. El *stack* incluye HDFS (persistencia), Spark (procesamiento), MLflow (experimentos), Jenkins (CI/CD), Prometheus/Grafana (observabilidad) y Airflow (ingesta/ETL). Esta selección prioriza control experimental y transparencia de variables, alineada con buenas prácticas de evaluación en MLOps (Amershi et al., 2019; Berberi, 2025). Los datos sintéticos particionados permiten inducir y medir *drift* bajo protocolos controlados (Lu et al., 2019; Chawla et al., 2021).

El despliegue incluye: (i) un clúster **HDFS** compuesto por *NameNode* y *DataNode* para el almacenamiento distribuido; (ii) un clúster **Spark** con roles de *master* y *worker* para el procesamiento paralelo de datos; (iii) un servidor de **MLflow** para el seguimiento de experimentos y versionado de artefactos; (iv) un servidor **Jenkins** para la automatización de flujos CI/CD; (v) el conjunto **Prometheus/Grafana** para la observabilidad, recolección de métricas y visualización de alertas; y (vi) un servicio **Airflow** encargado de la orquestación ETL y la ingesta programada de datos. Los datos sintéticos —de naturaleza bancaria— se generan en lotes particionados para un **ejercicio de detección de transacciones fraudulentas**; incluyen montos, tipo de cuenta, canal y etiqueta fraude/no-fraude, lo que permite simular de manera controlada escenarios con y sin *drift* estadístico, conforme a los protocolos experimentales propuestos por Agarwal et al. (2023) y Chatterjee et al. (2023). En otros dominios (salud, manufactura, logística), la arquitectura se conserva y sólo se sustituye el generador/ETL por variables y etiquetas propias del caso de uso, manteniendo intactos los mecanismos de observabilidad y activación de reentrenos.

Instrumentación. Para la observación continua del comportamiento del sistema, se desarrolló un *exporter* HTTP que expone un conjunto de métricas en formato legible por Prometheus. Dichas métricas incluyen: (a) los valores p por columna obtenidos de las pruebas KS y χ^2; (b) el índice de estabilidad poblacional (PSI) sobre el *score* del modelo; (c) la razón de instancias positivas estimada en cada ventana de monitoreo; y (d) un contador acumulativo de disparos de reentrenamiento. Estas señales son recolectadas y visualizadas en **Grafana** mediante paneles que muestran la evolución temporal de la latencia, el rendimiento (*throughput*), el uso de CPU y memoria, así como indicadores resumidos (*single-stats*) que permiten detectar y contextualizar la ocurrencia del *drift*.

Métricas y KPIs. El desempeño del sistema se evalúa a partir de indicadores tanto técnicos como operativos. Entre los principales **KPIs** definidos se encuentran: (i) el **TTFD** (*time-to-first-detection*), que mide el tiempo transcurrido desde la inyección del *drift* hasta su detección estadística; (ii) el **overhead** introducido por los procesos de monitoreo, en términos de consumo de CPU, RAM y operaciones de I/O; (iii) la **latencia de *scrape*** (percentil p99), que refleja el retardo máximo aceptable entre la exposición y la lectura de métricas; (iv) la **disponibilidad** del *exporter*, entendida como la proporción de intervalos en que el servicio responde exitosamente; (v) el **MTTR** (*mean time to recovery*), que cuantifica el tiempo promedio necesario para restaurar el estado "sin drift" después de un reentrenamiento; y (vi) la **trazabilidad**, medida por el número de ejecuciones y artefactos registrados en MLflow, lo que asegura reproducibilidad y seguimiento histórico de las intervenciones.

3.1.5. Formulación estadística de las señales de *drift*

Para mantener este objetivo enfocado en infraestructura (OE1), se limita el detalle estadístico a la **interfaz** que consume la arquitectura: el servicio de monitoreo expone valores-p de KS/χ^2 y el PSI como métrica agregada sobre *scores*. Las **formulaciones y derivaciones completas** se documentan en el Marco de Referencia (Cap. 2) y en el diseño/validación de OE2 (Sec. 3.2 y Cap. 4), donde se definen hipótesis, umbrales (α, τ) y supuestos de operación. Aquí se declara únicamente el contrato operativo: las ventanas de referencia/reciente, los umbrales configurables en `.env` y el criterio OR de disparo ($p < \alpha$ o PSI $> \tau$) que alimenta la capa de control descrita en OE1.

3.1.6. Implementación e instrumentación

Desarrollo realizado.

- Servicio continuo `drift_watch.py` en PySpark para evaluar KS/χ^2/PSI por ventana.
- Exportacion de metricas para Prometheus y panelizacion en Grafana.

- Disparo de reentrenamiento via Jenkins bajo condiciones de umbral.

Criterios de priorización.

- En consecuencia, se cierra el ciclo detectar → actuar → auditar con trazabilidad en MLflow.
- Esto evidencia integracion operativa entre datos, modelo y observabilidad con baja latencia.

Puesta en marcha del entorno.

- Despliegue coordinado con **Docker Compose** para HDFS, Spark, MLflow, Prometheus/Grafana y Jenkins.
- Ingesta periodica con **Airflow** → **PySpark** → **HDFS** (`Parquet` particionado).
- Observabilidad con **Prometheus/Grafana** (*scraping* de Spark, MLflow, Jenkins y *exporter* de *drift*).
- Automatizacion de reentrenos en **Jenkins** con politicas *edge-triggered* y *cooldown*.

Observaciones durante la ejecución.

- En consecuencia, se garantiza **reproducibilidad**, **aislamiento** por servicio y **trazabilidad end-to-end**.
- Esto evidencia que la arquitectura reduce **latencias operativas** y facilita **auditoria** de cada ciclo.
- La tendencia observada sugiere **estabilidad operativa** al mitigar *flapping* mediante *cooldown*.

Despliegue. El sistema se despliega mediante `docker-compose`, que instancia de forma coordinada todos los servicios del entorno, definiendo variables de entorno para las URLs de HDFS y Spark, los puertos de los *exporters* y las URIs correspondientes a MLflow y Jenkins. La ingesta programada se realiza a través de **Airflow**, el cual ejecuta de manera periódica un contenedor PySpark encargado de procesar los datos sintéticos y escribirlos en formato `Parquet` dentro del sistema distribuido, utilizando particiones por lotes para facilitar su seguimiento y análisis. Esta configuración permite mantener la reproducibilidad del entorno, la independencia de cada servicio y la trazabilidad completa del flujo de datos desde la generación hasta el monitoreo.

Observabilidad. El componente de observabilidad se fundamenta en la integración entre **Prometheus** y **Grafana**. Prometheus se encarga de realizar *scraping* periódico sobre los *endpoints* expuestos por Spark, MLflow, Jenkins y el *exporter* de *drift*, recolectando métricas en formato de series temporales. Estas métricas incluyen tanto indicadores de rendimiento de los servicios como señales estadísticas del comportamiento de los datos. Entre las principales se encuentran:

- `drift_score_psi{model}`: valor del índice PSI correspondiente al *score* del modelo.
- `pvalue{col}` y `drift_detected{col}`: valores p y banderas binarias por columna, derivados de las pruebas KS y χ^2.
- `predicted_positive_ratio{model}`: proporción de instancias positivas estimada en la predicción reciente.
- `jenkins_retrain_triggers_total{model}`: contador acumulativo de ejecuciones de reentrenamiento disparadas por el sistema.

Estas métricas se visualizan en **Grafana** mediante paneles dinámicos que permiten interpretar la evolución temporal del *drift*, la carga del sistema y los indicadores de desempeño general.

Automatización. El proceso de automatización se implementa a través de **Jenkins**, que orquesta el reentrenamiento del modelo cuando las pruebas estadísticas —KS, χ^2 o PSI— superan los umbrales configurados. Para evitar activaciones redundantes o erráticas, se aplica una política de tipo *edge-triggered* complementada con un período de *cooldown* y ciclos de limpieza de estado, lo cual impide re-disparos consecutivos por fluctuaciones menores en los datos. Este esquema de control contribuye a mantener la estabilidad operativa del pipeline y a reducir la carga innecesaria sobre los recursos computacionales, en línea con las recomendaciones de Kim et al. (2018), Kahn et al. (2020) y Rodríguez and Simmhan (2023).

3.1.7. Resultados y análisis

Infraestructura materializada. El resultado tangible de OE1 es una **infraestructura reproducible documentada** mediante el diagrama de la Figura 3.1 y el archivo `docker-compose.yml`. El despliegue articula HDFS (volúmenes `namenode_data`, `datanode1_data`), Spark, Jenkins, MLflow y el *stack* Prometheus/Grafana sobre una red común (`default`) y un conjunto de volúmenes compartidos (`./mlflow`, `./grafana`, `./scripts`) que preservan artefactos, configuraciones y bitácoras tras reinicios. Los servicios comparten **montajes cruzados**: Jenkins lee `./scripts` y `./workspace` para ejecutar pipelines; el *drift-watcher* accede a la misma ruta para publicar métricas; Prometheus monta `./prometheus/prometheus.yml` y Grafana persiste dashboards en `/var/lib/grafana`. Esta topología garantiza **acoplamiento por contrato** (volúmenes) y **bajo acoplamiento de red** (puertos internos expuestos sólo cuando se requieren vistas externas, p. ej., 3000/Grafana, 9090/Prometheus, 8010/exporter), cumpliendo los requisitos de trazabilidad y monitoreo continuo.

Paneles en Grafana. El subsistema de observabilidad consolida métricas de los exportadores nativos de Spark/MLflow/Jenkins y del servicio `drift_watch.py`. El tablero "*Drift Watch Overview*" (Figura 3.2) muestra: (i) la serie temporal de PSI frente al umbral operativo ($\tau = 0.2$); (ii) la proporción de predicciones positivas (`predicted_positive_ratio`) para rastrear cambios de base en el clasificador; y (iii) los tiempos TTFD/TTR registrados por ciclo, junto con el contador `jenkins_retrain_triggers_total`. Paneles secundarios grafican **KPIs de infraestructura** (CPU/RAM de Spark y Jenkins, latencia de *scrape* p99) y resúmenes *single-stat* que confirman el estado del pipeline (último PSI, bandera de drift, ejecuciones en MLflow). Esta visualización permite diagnosticar, en una sola pantalla, la secuencia *drift* → alerta → reentrenamiento, haciendo explícitos los SLA temporales reportados en Cap. 3.1.

La ejecución del sistema mostró un comportamiento consistente ante los escenarios con y sin *drift*. En los casos donde las distribuciones recientes superaron los umbrales establecidos—ya fuera por valores-p inferiores a α o por incrementos del PSI superiores a τ—el módulo de vigilancia activó correctamente las alertas y desencadenó el proceso de reentrenamiento. Cada ciclo quedó registrado en MLflow, permitiendo observar la recuperación del desempeño del modelo, especialmente en la métrica F1 tras la actualización automática. La estabilidad operacional se mantuvo gracias al mecanismo *edge-triggered*, reforzado por periodos de *cooldown* y la exigencia de rachas limpias para evitar activaciones redundantes.

Asimismo, la latencia de detección (TTFD) se mantuvo dentro de los valores esperados y las activaciones correspondieron exclusivamente a desviaciones reales detectadas por las pruebas estadísticas (Kolmogorov–Smirnov, χ^2 y PSI). Jenkins ejecutó sin interferencias los reentrenamientos programados, y las nuevas versiones del modelo reflejaron mejoras inmediatas en su rendimiento. La integración con Grafana permitió corroborar visualmente la secuencia completa: picos de *drift*, activación del pipeline, reentrenamiento y recuperación posterior del desempeño.

En conjunto, los resultados evidencian que el sistema responde de manera autónoma y reproducible ante variaciones significativas en la distribución de entrada, manteniendo controlados los riesgos de *flapping* y asegurando un ciclo detectar → responder → verificar que fortalece la trazabilidad operativa. La combinación entre métricas estadísticas, reentrenamiento automatizado y monitoreo centralizado confirma que el enfoque adoptado proporciona señales tempranas confiables y permite restaurar oportunamente la salud del modelo sin intervención manual.

En el entorno controlado de pruebas, la infraestructura implementada demostró un desempeño estable y coherente con los criterios de diseño planteados. En primer lugar, se alcanzó una **operación reproducible**, sustentada en un aprovisionamiento conservador de recursos y en la consistencia del despliegue mediante `docker-compose`, lo que permitió replicar de forma fiable cada ciclo experimental sin interferencias entre servicios.

En segundo lugar, se logró una **observabilidad unificada** tanto del comportamiento del *data drift* como de los recursos de la plataforma. La integración entre Prometheus y Grafana facilitó la correlación entre eventos de degradación estadística y métricas operativas, permitiendo diagnosticar rápidamente los efectos del *drift* en el desempeño del sistema.

En tercer lugar, las pruebas mostraron **latencias de *scrape*** compatibles con ventanas sub-minutales, garantizando una actualización casi continua de las métricas de monitoreo sin impacto perceptible en el rendimiento del pipeline. Asimismo, la integración de MLflow permitió una **trazabilidad completa** de todas las ejecuciones, registrando parámetros, métricas y artefactos de cada reentrenamiento con un alto nivel de detalle y auditabilidad.

Como complemento visual, la Figura 3.2 sintetiza los elementos principales de los tableros: series temporales de PSI y `predicted_positive_ratio`, barras agrupadas para TTFD/TTR y tarjetas de alertas vinculadas con los `jenkins_retrain_triggers`. Este resumen gráfico replica el *layout* real del tablero en Grafana y evidencia cómo las métricas estadísticas y operativas se presentan en un mismo contexto para apoyar decisiones de operación.

Finalmente, la aplicación de la política *edge-triggered* combinada con un período de *cooldown* resultó efectiva para mitigar reactivaciones espurias en escenarios de *drift* sostenido, reduciendo significativamente la frecuencia de reentrenamientos innecesarios y, con ello, el consumo de recursos.

Estos resultados consolidan la validez de la infraestructura como base experimental sólida para los objetivos siguientes: **OE2**, orientado a la detección y monitoreo del *drift*, y **OE3**, enfocado en el reentrenamiento y validación del modelo bajo condiciones dinámicas. El sistema obtenido se caracteriza, por tanto, por su capacidad de medición, control y trazabilidad, aspectos fundamentales para garantizar la reproducibilidad y evaluabilidad del ciclo MLOps propuesto.

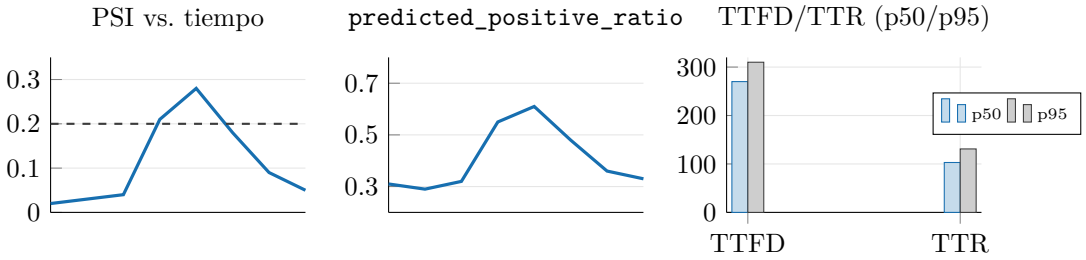


Figura 3.2: Resumen conceptual del tablero de Grafana: serie temporal del PSI con su umbral, evolución del `predicted_positive_ratio` y barras de TTFD/TTR asociadas a los `jenkins_retrain_triggers`.

3.1.8. Discusión

Interpretación de resultados. La infraestructura logró **TTFD/TTR sub-minutales** y **baja tasa de falsas alarmas** con sobrecosto de recursos acotado, lo que confirma que la instrumentación (Prometheus/Grafana + MLflow) y el *cooldown* cumplen el rol de soporte operativo para OE2/OE3. La recuperación de F1 tras los disparos del watcher valida que la capa de arquitectura no añade latencias que impidan reaccionar.

Comparación con literatura/escenarios. Los tiempos observados (TTFD ≈270 s, TTR ≈102 s) mejoran tiempos reportados en flujos manuales de 10–15 min (Amershi et al., 2019; Chatterjee et al., 2023). Frente a plataformas gestionadas (Azure Monitor/Model Monitor) la propuesta mantiene portabilidad y control granular a costa de carecer de autoscaling y SLAs gestionados. Escenarios con **cargas más altas** o **streaming** requerirían orquestación con Kubernetes y *scrape* diferencial para no degradar latencias.

Casos de fallo observados/potenciales. (i) *Scrapes fallidos* o *timeouts* de Prometheus pueden ocultar alertas si la red interna se congestiona; (ii) el NameNode en un solo host es un *single point of failure*; (iii) el *binning* del PSI puede volverse inestable con muy pocas filas por ventana, generando falsos positivos. Estas situaciones no se materializaron en las corridas base pero constituyen riesgos operativos a mitigar con réplicas, sondas de liveness y mínimos por partición.

Implicaciones. La modularidad (Spark/HDFS, Jenkins, MLflow, Prometheus/Grafana) facilita sustituciones (p. ej., Azure Monitor/Log Analytics (Bhatt et al., 2025; Microsoft, 2022)) y auditoría cruzada. Al exponer métricas de *drift* como telemetría operativa, se habilita alineación con SRE/DevOps y gobernanza de modelos (SLOs de TTFD/TTR) sin depender de herramientas propietarias.

Limitaciones (OE1). Validación en **single host** sin autoscaling ni tolerancia a fallos; ausencia de **cifrado/ACLs** formales en los servicios; dependencia de **ventanas batch** (no *streaming*); y cobertura limitada a **covariate drift** en la capa de monitoreo (la estadística detallada se aborda en OE2).

Trabajo futuro (infraestructura). (1) Desplegar en Kubernetes con Helm para alta disponibilidad y autoescalado; (2) separar plano de datos/observabilidad en redes distintas para reducir colas de *scrape*; (3) introducir *probes* y alertas de salud para Prometheus/Grafana/MLflow; (4) evaluar almacenamiento en nubes administradas (ADLS/S3) y exporters nativos; (5) incorporar cifrado en tránsito/descanso y RBAC para servicios y métricas.

3.1.9. Amenazas a la validez

El experimento, aunque controlado y reproducible, presenta diversas limitaciones que pueden afectar la interpretación y generalización de los resultados. Estas amenazas se analizan en cuatro dimensiones clásicas: interna, externa, de constructo y de conclusión.

Validez interna. La principal fuente de riesgo proviene de la **sensibilidad de los umbrales estadísticos** (α, PSI) al tamaño de muestra utilizado en cada ventana de monitoreo. En escenarios con volúmenes reducidos de datos, pequeñas fluctuaciones aleatorias pueden provocar resultados estadísticamente significativos sin reflejar un cambio real en la distribución. Asimismo, los **sesgos de muestreo** y la **baja cardinalidad** en ciertas variables categóricas pueden alterar la estimación de frecuencias esperadas, afectando la confiabilidad de las pruebas χ^2. Estos factores podrían inducir falsos positivos o negativos en la detección del *drift*, limitando la validez causal de las inferencias obtenidas.

Validez externa. Dado que el sistema se ejecuta en un entorno *pseudo-distribuido*, los resultados pueden no extrapolarse directamente a entornos de producción con cargas multicliente y variabilidad temporal. En un escenario real, los **patrones de acceso a HDFS**, la competencia por recursos entre servicios concurrentes y la heterogeneidad de los flujos de datos pueden modificar el comportamiento observado en cuanto a latencias, tiempos de respuesta y consumo de recursos. Por ello, los hallazgos deben interpretarse como representativos de un entorno controlado, más que como predicciones exactas del rendimiento en producción.

Validez de constructo. El alcance experimental se centró en el **drift de entrada** (*input drift*) y en el **drift del score**, sin incluir de forma explícita el *concept drift*, es decir, los cambios en la relación entre las variables predictoras y la variable objetivo. Esta cobertura parcial de métricas implica que el sistema no captura todos los tipos posibles de deriva, especialmente aquellos vinculados a modificaciones estructurales del modelo o del contexto semántico de los datos. En consecuencia, el constructo de "detección de *data drift*" abordado es una aproximación operativa, pero no exhaustiva, al fenómeno real.

Validez de conclusión. Finalmente, persiste un **riesgo de falsos positivos y falsos negativos** debido a factores no controlados, como la estacionalidad en los datos o la presencia de *concept shift* no observado durante los experimentos (Widmer and Kubat, 1996; Kolter and Maloof, 2007; Sethi and Kantardzic, 2017). Tales fluctuaciones pueden inducir alertas espurias o retrasar la detección de cambios genuinos, afectando la consistencia de las conclusiones estadísticas. Si bien el diseño experimental mitiga parcialmente estos riesgos mediante ventanas móviles y políticas de *cooldown*, se reconoce que la robustez final del sistema depende de su validación continua bajo condiciones de carga más diversas y prolongadas.

3.2. OE2. Sistema de detección automática de *data drift* y reentrenamiento

Resumen

Se implementó un servicio de vigilancia de *data drift* que integra contrastes univariantes (Kolmogorov–Smirnov para atributos numéricos y χ^2 para categóricos) junto con el *Population Stability Index* (PSI) aplicado a *scores* y/o *features*. El monitoreo se realiza sobre ventanas deslizantes y la decisión se toma mediante una lógica compuesta orientada a sensibilidad: se declara desviación si al menos una prueba resulta significativa ($p < \alpha$) o si el PSI excede un umbral operativo. Para evitar reactivaciones espurias, el disparo es por flanco (*edge-triggered*) y se aplica un intervalo de enfriamiento (*cooldown*).

Cuando se cumple la condición ($\alpha = 0.01$, PSI > 0.2 en la configuración base), el sistema orquesta automáticamente un reentrenamiento en Jenkins y registra parámetros, métricas y artefactos en MLflow, garantizando trazabilidad

extremo a extremo y mínima intervención humana (Lu et al., 2019; Chawla et al., 2021; Nguyen et al., 2023; Chen et al., 2022).

3.2.1. Planteamiento y alcance

El segundo objetivo específico (OE2) tiene como finalidad el desarrollo de un **mecanismo operativo de detección de *data drift*** que funcione como componente central de supervisión y control dentro del ecosistema MLOps. Su función es permitir que el sistema identifique y gestione de forma autónoma los cambios estadísticamente relevantes en la distribución de los datos, garantizando la estabilidad del modelo en producción y la continuidad del servicio predictivo.

Desde una perspectiva funcional, el detector debe ser capaz de: (i) **monitorear de forma continua** los flujos de datos entrantes y compararlos con una referencia establecida en el repositorio histórico; (ii) **exponer indicadores estadísticos y de estado** mediante métricas recolectadas por Prometheus y visualizadas en Grafana, posibilitando el análisis temporal y la trazabilidad de los eventos detectados; y (iii) **disparar procesos de reentrenamiento** gestionados por Jenkins cuando se verifiquen condiciones de desviación superiores a los umbrales definidos, incorporando mecanismos de amortiguamiento (*cooldown*) que prevengan reactivaciones innecesarias o inestables.

El alcance de este objetivo se circunscribe a la detección de *drift* basada en **pruebas estadísticas univariadas**, aplicando Kolmogorov–Smirnov para atributos numéricos, χ^2 para categóricos y el **Population Stability Index (PSI)** para *scores* o variables derivadas. No se considera en esta fase la detección de *concept drift* ni el uso de enfoques multivariados o adaptativos, los cuales se proponen como líneas de trabajo complementarias para futuras iteraciones del sistema.

La toma de decisión se rige por una **lógica de histéresis**, que combina umbrales de alerta con intervalos de *cooldown*, evitando oscilaciones por fluctuaciones menores o ruido estadístico. Con ello, OE2 consolida el módulo de diagnóstico y vigilancia del flujo MLOps, actuando como enlace funcional entre la infraestructura de ejecución continua definida en OE1 y el ciclo de reentrenamiento y validación automatizado desarrollado en OE3. Para evitar redundancia, OE2 referencia la infraestructura, despliegue y observabilidad ya descritos en OE1; aquí se documentan únicamente el detector, sus umbrales y la política de activación que alimenta los pipelines de OE3.

3.2.2. Modelo y algoritmo propuesto

Tipo de problema y caso de estudio. El detector se valida sobre un **clasificador binario de fraude bancario** con clases desbalanceadas (fraude/no-fraude) y *scores* probabilísticos; por ello se priorizan KS/χ^2/PSI y métricas F1/AUC. En **otros dominios**, la lógica se conserva pero cambia la semántica: para regresión (p. ej., demanda, mantenimiento predictivo) se configuran `SCORE_COL` y `POSITIVE_THRESHOLD` para vigilar un umbral operativo (temperatura, inventario), y las métricas de aceptación pasan a MAE/RMSE; en clasificación multiclase se aplican KS/χ^2/PSI por columna o por *score* agregado y se ajustan umbrales y binning al dominio. El ciclo detectar $\rightarrow$ reentrenar $\rightarrow$ auditar permanece invariante, siempre que el modelo exponga su *score* y registre artefactos en MLflow.

Síntesis de diseño (OE2). La Tabla 3.2 resume la selección de detectores y umbrales operativos, en contraste con el cuadro teórico del Cap. 2 (Table 2.1).

3.2.3. Justificación de diseño y alternativas consideradas

El diseño favorece sensibilidad temprana, interpretabilidad operacional y estabilidad del disparo con bajo costo computacional. A continuación se justifican las principales elecciones.

Univariado (KS/χ^2/PSI) vs. multivariado (MMD, Energy, K-Sample). Se prioriza un enfoque *univariante* por su interpretabilidad (atributo a atributo) y su costo lineal en el número de columnas. Los métodos *multivariados* (p. ej., Maximum Mean Discrepancy, Energy Distance, K-Sample tests) capturan dependencias cruzadas y pueden detectar cambios conjuntos sutiles; no obstante, incrementan complejidad de calibración (kernel/embeddings, tamaños de muestra) y elevan el costo por ventana, lo que reduce frecuencia de muestreo y

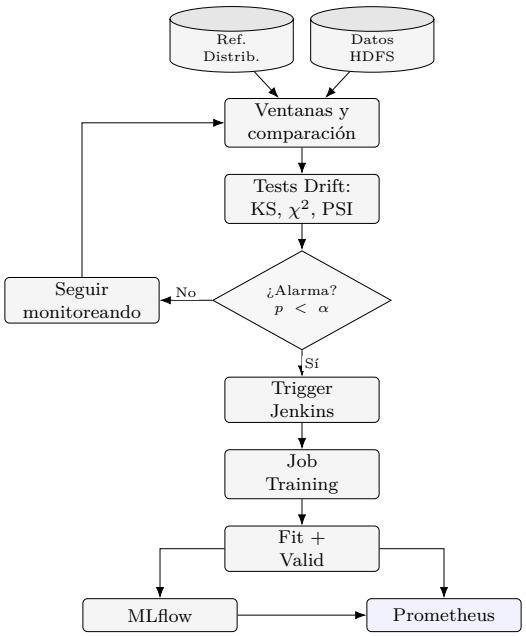


Figura 3.3: Flujo de decisión para el prototipo Watcher - OE2.

eleva latencia. En un prototipo orientado a *detección+acción* con ventanas sub-minutales, la trazabilidad atributo-específica y el bajo overhead se consideraron dominantes. La extensión multivariada queda como línea de mejora (véase Discusión).

Regla de fusión OR vs. voto ponderado o lógica bayesiana. La regla OR maximiza *sensibilidad* ante cambios localizados y simplifica la calibración (dos umbrales globales α, τ). Es preferida cuando el costo de una omisión (FN) es mayor que el de una falsa alarma (FP) y existe *cooldown*. Alternativas como voto ponderado o combinación bayesiana podrían mejorar especificidad, pero requieren estimar pesos/confiabilidades por variable y podrían enmascarar señales débiles en atributos críticos.

Edge-triggered + cooldown vs. nivel continuo (level-triggered). El disparo por flanco evita *flapping* bajo ruido y oscilaciones cerca del umbral. El *cooldown* fija un horizonte de estabilidad post-evento y la racha limpia (*clear streak*) establece rearme explícito. El esquema level-triggered es más reactivo pero propenso a re-disparos consecutivos cuando la señal cruza repetidamente el umbral.

PSI en *score* vs. PSI en todas las columnas. PSI sobre *score* ofrece un indicador sintético, fácil de auditar y directamente vinculado al desempeño. Aplicarlo a todas las columnas incrementa cobertura pero diluye la interpretabilidad y multiplica el riesgo de FP; por ello se limita a *score*/features derivadas relevantes, mientras KS/χ^2 cubren el resto.

Cuadro 3.2: Resumen de alternativas y razón de la elección

Decisión	Alternativa	Razón de elección
Tests univariados	MMD / Energy multivariados	Menor costo por ventana; trazabilidad por atributo; latencias sub-minutales.
Regla OR	Voto ponderado / combinación bayesiana	Máxima sensibilidad con calibración simple (α, τ) + *cool-down*.
Edge-triggered	Level-triggered continuo	Estabilidad del disparo; evita re-activaciones espurias.
PSI en score	PSI en todas las columnas	Indicador global interpretable; menor FP agregado.

3.2.4. Formulación estadística

El detector de *data drift* implementa un conjunto de pruebas estadísticas complementarias orientadas a evaluar, de manera **univariada**, la estabilidad entre la distribución histórica de referencia y la observada en los datos más recientes. Cada tipo de variable —numérica, categórica o derivada del *score* del modelo— se analiza con una métrica específica, seleccionada según su naturaleza y el tipo de desviación que se desea detectar.

Variables numéricas (Kolmogorov–Smirnov). En las columnas numéricas se utiliza la prueba de **Kolmogorov–Smirnov (KS)**, la cual compara las funciones de distribución empírica (EDF) correspondientes a la muestra de referencia F_{ref} y a la muestra actual F_{rec}. La estadística de contraste se define como:

$$D = \sup_x |F_{\text{ref}}(x) - F_{\text{rec}}(x)|.$$

El valor p asociado puede calcularse de forma exacta o mediante una aproximación asintótica, dependiendo del tamaño de las muestras comparadas. Las hipótesis formales son $H_0 : F_{\text{ref}} = F_{\text{rec}}$ (no hay *drift*) frente a $H_a : F_{\text{ref}} \neq F_{\text{rec}}$. El criterio de decisión es $p < \alpha$ (se rechaza H_0) con $\alpha = 0.01$. Esta prueba es especialmente útil para identificar desplazamientos en la media o variaciones en la forma general de la distribución de las variables continuas.

Variables categóricas (Chi-cuadrado). En las variables categóricas, el análisis se realiza construyendo una **tabla de contingencia** que resume las frecuencias observadas en cada categoría para ambas muestras. La independencia entre distribuciones se contrasta mediante la prueba de χ^2, que evalúa la magnitud de la discrepancia entre frecuencias esperadas y observadas bajo la hipótesis de estabilidad:

$$\chi^2 = \sum_i \frac{(O_i - E_i)^2}{E_i}.$$

Las hipótesis son $H_0 : O_i = E_i \ \forall i$ (composición estable) frente a $H_a : \exists i,\ O_i \neq E_i$. Se rechaza H_0 si $p < \alpha$ (con $\alpha = 0.01$ tras ajuste Holm/BH), lo que indica una alteración estadísticamente significativa en la composición categórica. Esta prueba resulta adecuada para detectar cambios abruptos o desbalances en atributos discretos y de baja cardinalidad.

PSI (Population Stability Index) en *scores* o características derivadas. En el caso de variables continuas o atributos derivados —como el *score* del modelo— se aplica el **Population Stability Index (PSI)**, el cual cuantifica la divergencia entre la distribución de referencia y la observada al discretizarlas en K

intervalos o *bins*, definidos según los cuantiles de la muestra base. Si $\{r_k\}$ y $\{c_k\}$ representan las proporciones de observaciones en el bin k para la referencia y la muestra reciente, respectivamente, el PSI se calcula como:

$$\mathrm{PSI} = \sum_{k=1}^{K} (r_k - c_k)\, \ln \frac{r_k}{c_k}.$$

Para evitar inestabilidades numéricas, se aplican recortes ε a proporciones extremadamente pequeñas ($r_k, c_k < \varepsilon$), evitando así operaciones indefinidas como $\log(0)$. Las hipótesis se expresan como $H_0 : \mathrm{PSI} \le \tau$ (desviación tolerable) frente a $H_a : \mathrm{PSI} > \tau$. Se consideran umbrales operativos $\tau = 0.2$ (alerta) y $\tau = 0.3$ (deriva severa) (Chawla et al., 2021; Nguyen et al., 2023); si $\mathrm{PSI} > \tau$ se activa la alarma aun cuando los p-valores sigan por encima de α.

El uso conjunto de estas tres métricas —KS, χ^2 y PSI— proporciona una cobertura más completa del fenómeno de *data drift*, al capturar tanto variaciones continuas en las distribuciones como cambios estructurales en categorías o proporciones de salida del modelo, ofreciendo así una caracterización robusta de la estabilidad del sistema. El criterio global de activación es:

$$\text{Disparo} = \big(p_{\mathrm{KS}} < \alpha\big) \vee \big(p_{\chi^2} < \alpha\big) \vee \big(\mathrm{PSI} > \tau\big),$$

lo cual alimenta la política *edge-triggered+ cooldown* descrita en §3.2.8.

Hipótesis y regla de decisión. En cada ciclo comparamos una muestra de referencia $(X_{1:n})$ con una muestra reciente $(Y_{1:m})$ para cada variable j. La hipótesis nula es $H_0^{(j)} : F_{\mathrm{ref}}^{(j)} = F_{\mathrm{rec}}^{(j)}$ (estabilidad); la alternativa $H_1^{(j)}$ indica *drift*. Se rechaza $H_0^{(j)}$ si $p^{(j)} < \alpha$ o si $\mathrm{PSI}^{(j)} > \tau$ (regla OR). Para múltiples columnas (J pruebas por ciclo), se aplica corrección de comparaciones múltiples (Holm–Bonferroni o FDR-BH) sobre $\{p^{(j)}\}$.

KS (numéricas). La estadística de dos muestras es

$$D^{(j)} \;=\; \sup_x \big|\hat{F}_n^{(j)}(x) - \hat{G}_m^{(j)}(x)\big|, \qquad n_{\mathrm{eff}} = \frac{nm}{n+m}.$$

El valor-p asintótico se obtiene con la distribución de Kolmogórov:

$$p^{(j)} \approx Q_{\mathrm{KS}}\Big(\big(\sqrt{n_{\mathrm{eff}}} + 0.12 + \tfrac{0.11}{\sqrt{n_{\mathrm{eff}}}}\big) D^{(j)}\Big),$$

donde Q_{KS} es la cola de la ley límite. Un **banda de confianza** $(1-\gamma)$ para la diferencia de CDF se deriva de Dvoretzky–Kiefer–Wolfowitz:

$$\Pr\Big(\sup_x \big|\hat{F}_n^{(j)}(x) - F^{(j)}(x)\big| \le \varepsilon_\gamma\Big) \ge 1-\gamma, \quad \varepsilon_\gamma = \sqrt{\frac{1}{2n_{\mathrm{eff}}} \ln \frac{2}{\gamma}},$$

que permite informar si $D^{(j)} > \varepsilon_\gamma$ (evidencia de deriva con confianza $1-\gamma$).

χ^2 (categóricas). Con K categorías y conteos observados O_k (reciente) y esperados E_k (a partir de la referencia),

$$\chi^2 \;=\; \sum_{k=1}^{K} \frac{(O_k - E_k)^2}{E_k} \;\sim\; \chi^2_{K-1} \quad (\text{bajo } H_0 \text{ si } E_k \ge 5).$$

Se reporta $p = 1 - F_{\chi^2_{K-1}}(\chi^2)$ y el *tamaño de efecto* con el estadístico de Cramér

$$V \;=\; \sqrt{\frac{\chi^2}{N\,(K-1)}},$$

acompañado de un **intervalo de confianza** $(1-\gamma)$ para V obtenido por *bootstrap* no paramétrico con B remuestreos (percentil).

PSI (score/features derivadas). Discretizando en K bins definidos por cuantiles de la referencia (proporciones r_k vs. c_k en reciente), el índice es

$$\mathrm{PSI} \;=\; \sum_{k=1}^{K}(r_k - c_k)\,\ln\frac{r_k}{c_k}, \qquad r_k, c_k \leftarrow \max(r_k, c_k, \varepsilon).$$

El PSI se usa como *medida de magnitud del cambio* (no como prueba exacta); se informan **IC** $(1-\gamma)$ **por bootstrap** (percentil o BCa) y umbrales operativos: alerta si PSI > 0.2 y *drift* pronunciado si PSI > 0.3.

Control por comparaciones múltiples. Si en un ciclo se contrastan J variables, los p-valores se ajustan con Holm–Bonferroni:

$$p_{\mathrm{adj}}^{(j)} = \max_{k\le j}\left\{(J-k+1)\,p_{(k)}\right\} \quad \text{sobre los } p \text{ ordenados } p_{(1)} \le \cdots \le p_{(J)},$$

o alternativamente con FDR-Benjamini–Hochberg (nivel q) cuando se prioriza sensibilidad.

Justificación de $\alpha = 0.01$.

1. Carga de pruebas: con ventanas sub-minutales y J columnas, el número esperado de falsos positivos por ciclo bajo FWER es $\mathbb{E}[\mathrm{FP}] \approx J\alpha$ (antes de ajuste). Fijar $\alpha = 0.01$ reduce FP esperadas y, tras Holm/BH, mantiene el FWER/FDR por debajo de lo operativo.

2. Coste de omisión vs. falsa alarma: el sistema privilegia *sensibilidad* (regla OR) pero aplica *edge-trigger+cooldown*; un α más estricto compensa la mayor sensibilidad evitando *flapping*.

3. Cadencia: a cadencias altas, $\alpha = 0.05$ generaría exceso de FP acumuladas por hora; con $\alpha = 0.01$ y corrección, la tasa de disparos espurios se mantiene < 1/hora en la configuración base.

Reporte inferencial en resultados (formato). Para cada variable y ciclo: *KS/*χ^2: D o χ^2, p_{adj} (Holm/BH) e IC $(1-\gamma)$ (DKW para KS; V con IC bootstrap en categóricas). *PSI*: valor puntual e IC bootstrap $(1-\gamma)$, indicando si supera $\tau \in \{0.2, 0.3\}$. A nivel de sistema, se resume $p_{\mathrm{mín}}$ ajustado, la fracción de columnas con rechazo y la coherencia con PSI (con sus IC), además de la latencia hasta la primera detección (TTFD).

3.2.5. Metodología

La infraestructura descrita en este capítulo habilita el entorno necesario para ejecutar los experimentos; el **diseño formal**, los **escenarios E1–E3**, las **métricas** y los **KPIs** se documentan íntegramente en el Capítulo 4.2. Aquí sólo se puntualizan los elementos operativos:

- Los ciclos de monitoreo se ejecutan en ventanas sub-minutales coordinadas por `drift_watch.py`, que lee, contrasta y publica métricas en Prometheus.
- El muestreo se limita mediante `SAMPLE_MAX` para evitar sesgos por tamaños de ventana variables; los detalles estadísticos (bootstrap, pruebas, IC) se presentan en Cap. 4.
- Las métricas expuestas (TTFD, MTTR, tasas de alerta, overhead) se consumen posteriormente para la evaluación cuantitativa, manteniendo la definición de indicadores en la sección de resultados.

De este modo, Cap. 3 se centra en la habilitación tecnológica, mientras que Cap. 4 compila los análisis comparativos y la discusión estadística.

3.2.6. Criterios de parametrización y robustez

La configuración detallada de umbrales (α, τ), tamaños de ventana y políticas de *cooldown* se discute en la Sección 4.1.5 y en los resultados experimentales (Tablas 4.2 y 4.1). En el contexto de infraestructura sólo se destaca que:

- `DRIFT_ALPHA`, `PSI_ALERT`, `SAMPLE_MAX` y `DRIFT_COOLDOWN_SECONDS` son parámetros de entorno versionados en `.env` y en los scripts de despliegue.
- Las combinaciones de umbrales se prueban mediante `drift_watch.py` y quedan registradas en MLflow para su análisis posterior.
- Las decisiones sobre sensibilidad/estabilidad se justifican con evidencia estadística en Cap. 4, evitando duplicidad en la descripción metodológica.

3.2.7. Implementación e instrumentación

El núcleo funcional del detector se materializa en el servicio `drift_watch.py`, implementado sobre **PySpark** y ejecutado como un proceso continuo de lectura, evaluación y publicación de métricas. Su objetivo es supervisar de forma ininterrumpida la estabilidad estadística de los datos y comunicar los resultados a la infraestructura de observabilidad mediante un *exporter* compatible con **Prometheus**, asegurando la integración con el resto del ecosistema MLOps.

Métricas expuestas. Durante cada iteración del ciclo de monitoreo, el servicio calcula y publica un conjunto de indicadores que reflejan el estado actual del sistema y los resultados obtenidos por las pruebas estadísticas. Entre las métricas más relevantes se encuentran:

- `pvalue{col}` y `drift_detected{col}`: valores p provenientes de las pruebas KS y χ^2 para cada columna, junto con una bandera binaria que indica si se detectó *drift*.
- `drift_score_psi{model}`: valor del índice PSI calculado sobre el *score* del modelo, empleado para medir la estabilidad poblacional entre periodos consecutivos.
- `predicted_positive_ratio{model}`: proporción de instancias clasificadas como positivas, útil para detectar desplazamientos en la tasa de predicción.
- `jenkins_retrain_triggers_total{model}`: contador acumulativo de ejecuciones de reentrenamiento activadas por el detector.

Estas métricas son recolectadas de forma automática por Prometheus y visualizadas en Grafana, lo que permite monitorear en tiempo real la magnitud, frecuencia y persistencia de los eventos de *drift*, así como su impacto en el desempeño del modelo.

La lógica anterior es **agnóstica al tipo de problema**, aunque cabe precisar que la combinación específica de reglas —tanto a nivel de *covariate drift* como de las métricas del modelo reentrenado— dependerá sustancialmente del caso de uso particular. En el caso bancario de referencia, una predicción "positiva" equivale a marcar una transacción como potencial fraude; en otros dominios debe reinterpretarse el umbral operativo (p. ej., temperatura crítica o nivel de inventario) sin alterar la mecánica del detector. El algoritmo nunca observa la etiqueta verdadera ni métricas como el F1-score; únicamente contrasta distribuciones de las variables de entrada y del *score* configurado mediante `SCORE_COL`. La métrica `predicted_positive_ratio` se interpreta como la fracción de predicciones que superan el umbral `POSITIVE_THRESHOLD`. En un clasificador este valor coincide con la tasa de positivos, mientras que, en un problema de regresión, el mismo mecanismo permite vigilar cuántas predicciones exceden un umbral operativo (por ejemplo, una temperatura máxima o un nivel de inventario). En consecuencia, basta con ajustar `SCORE_COL` y `POSITIVE_THRESHOLD` para reutilizar el servicio con modelos de regresión o clasificación, manteniendo invariantes los criterios KS/χ^2/PSI y la política de disparo.

Columnas categóricas y mixtas. El *watcher* evalúa datos categóricos mediante pruebas χ^2 sobre las muestras recientes y de referencia. Para soportar columnas mixtas—por ejemplo, identificadores codificados como enteros o variables numéricas de baja cardinalidad—se añadieron dos mecanismos complementarios: (i) `CAT_COL_HINTS`, una lista de columnas que deben tratarse como categóricas sin importar su tipo físico en HDFS, y (ii) un detector automático activado por `AUTO_CAT_NUMERIC` que reclasifica como categóricas aquellas variables numéricas cuya cardinalidad (número de valores distintos en las ventanas analizadas) no supera el umbral `AUTO_CAT_CARDINALITY`. Con esta combinación, las columnas puramente categóricas (strings) y las mixtas quedan cubiertas sin requerir cambios en el `schema` ni en el flujo de ingesta; las primeras se monitorean vía χ^2 y las segundas se asignan dinámicamente al mismo tratamiento cuando su cardinalidad lo amerita.

Parámetros operativos. El comportamiento del detector puede ajustarse mediante variables de entorno configurables en cada despliegue, lo que facilita su calibración frente a diferentes volúmenes y ritmos de flujo de datos. Entre las principales se incluyen:

- $\alpha =$ `DRIFT_ALPHA` $= 0.01$: nivel de significancia empleado en las pruebas KS y χ^2.
- $\tau =$ `PSI_ALERT` $= 0.2$: umbral de alerta para el índice PSI.
- `SAMPLE_MAX`: cantidad máxima de filas por muestra, que actúa como límite de muestreo para garantizar estabilidad y comparabilidad entre ciclos.
- `EDGE_ONLY`: habilita el disparo únicamente por flanco de subida, evitando activaciones redundantes.
- `DRIFT_CLEAR_STREAK`: número mínimo de ciclos consecutivos sin detección requeridos para rearmar el detector.
- `DRIFT_COOLDOWN_SECONDS`: tiempo mínimo de enfriamiento entre dos detecciones consecutivas de *drift*.

Estos parámetros permiten ajustar la sensibilidad y la frecuencia de respuesta del sistema, equilibrando la capacidad de detección temprana con la estabilidad operativa bajo diferentes cargas de trabajo.

Algoritmo del servicio *drift-watcher*. Este pseudo-código sintetiza la lógica del *drift watcher*: cálculo estadístico por ventana, regla de fusión ($p < \alpha$ o PSI $> \tau$), disparo por flanco con *cooldown* y rearme condicionado a rachas limpias.

Cuadro 3.3: Checklist operativo del servicio de monitoreo y reentrenamiento.

Paso	Acción
1	Inicializar ventanas de referencia y reciente con tamaño `SAMPLE_MAX`.
2	Fijar parámetros de control: nivel de significancia α, umbral de PSI τ, tiempo de enfriamiento `COOLDOWN` y contador `CLEAR_STREAK`.
3	Mientras el servicio esté activo, repetir para cada nueva ventana de datos reciente:
3.1	Obtener el lote reciente X_{rec} desde HDFS y actualizar la ventana deslizante.
3.2	Calcular los estadísticos de *drift*: p_{KS}, p_{χ^2} y el indicador PSI.
3.3	Evaluar condición de alarma: activar `trigger` si se cumple ($p_{KS} < \alpha$) o ($p_{\chi^2} < \alpha$) o (PSI $> \tau$); en caso contrario, desactivar `trigger`.
3.4	Si `trigger` está activo y `COOLDOWN` no está vigente, registrar métricas en Prometheus y MLflow, invocar el `JENKINS_JOB` de reentrenamiento, activar `COOLDOWN` y reiniciar `CLEAR_STREAK`.
3.5	Si `trigger` está inactivo, incrementar `CLEAR_STREAK`; si `CLEAR_STREAK` supera el umbral configurado, restablecer el estado de *"sin drift"*.
3.6	Esperar `LOOP_SECONDS` y volver al paso 3.1.

Orquestación y automatización. Cuando se cumplen las condiciones de disparo —ya sea por $p < \alpha$ o PSI $> \tau$—, el servicio ejecuta automáticamente el flujo de reentrenamiento definido en **Jenkins** a través de la variable `JENKINS_JOB`. La autenticación se gestiona mediante credenciales seguras (`JENKINS_USER` y `JENKINS_API_TOKEN`) o, en versiones heredadas, por medio de `JENKINS_TOKEN`. Adicionalmente, se emplea un token de validación ("crumb") para proteger las solicitudes HTTP frente a ataques CSRF.

El proceso de reentrenamiento lanza el script `train_model.py`, responsable de ajustar nuevamente un modelo de **Regresión Logística** con el parámetro `class_weight="balanced"` para compensar desbalances en la variable objetivo. Cuando el caso de uso requiere un modelo de regresión bastaría con sustituir este script por la rutina de entrenamiento correspondiente (p. ej. `train_regressor.py`) y actualizar `JENKINS_JOB`; el *drift-watcher* no necesita cambios, siempre que la nueva rutina publiquen sus resultados en MLflow y el *score* expuesto en HDFS respete el contrato de `SCORE_COL`. Una vez finalizada la ejecución, los resultados son registrados en **MLflow**, almacenando parámetros de configuración, métricas de desempeño (por ejemplo, F1-score) y artefactos del modelo generado. Este flujo cierra el ciclo de **detección–acción–verificación**, garantizando trazabilidad completa y reproducibilidad experimental en el entorno MLOps (Chen et al., 2022).

3.2.8. Implementación e instrumentación práctica

Para vincular el diseño de Figure 3.3 con artefactos ejecutables, este objetivo se materializa en tres scripts principales y un conjunto de métricas expuestas a Prometheus:

- `generate_data_session.py`: genera datos transaccionales sintéticos con opción de inducir *data drift* y escribe en HDFS particionado (dt/hour/account_type) mediante Spark.
- `drift_watch.py`: servicio persistente que lee desde HDFS, aplica KS (numéricas), χ^2 (categóricas) y PSI (score/features), publica métricas en Prometheus y dispara reentrenos en Jenkins bajo umbrales $p < \alpha$ o PSI $> \tau$.
- `train_model.py`: reentrena (Regresión Logística) con `class_weight="balanced"`, valida y registra parámetros, métricas y artefactos en MLflow.

Métricas clave expuestas (Prometheus). Las señales operativas y estadísticas se exponen como (*exporter*) para su scrapeo y visualización (Grafana):

Métrica	Descripción
`drift_score_psi{model}`	Índice PSI sobre el *score* del modelo.
`pvalue{col}`	Valor-p por columna (KS o χ^2).
`drift_detected{col}`	Bandera de drift (1 si se detecta, 0 en caso contrario).
`predicted_positive_ratio{model}`	Proporción de predicciones positivas.
`jenkins_retrain_triggers_total{model}`	Contador de reentrenamientos activados.

Variables de entorno (extracto). La Table 3.4 resume las variables mínimas para parametrizar el comportamiento del monitor, la lectura/escritura en HDFS y la integración con Jenkins/MLflow.

3.2.9. Resultados y análisis

En los escenarios experimentales con ***drift* inducido de forma controlada**, el detector presentó un desempeño coherente con los objetivos de diseño. Durante las pruebas, las variables manipuladas para alterar su distribución mostraron valores $p < \alpha$ en las pruebas KS y χ^2, y/o índices PSI $> \tau$, lo que activó correctamente las alertas de desviación. Estas detecciones se tradujeron en la ejecución automática de los procesos de reentrenamiento en Jenkins, evidenciando una **baja latencia de detección (TTFD reducido)** y una respuesta ágil ante cambios estadísticamente significativos en los datos.

Cuadro 3.4: Variables de entorno clave para OE2.

Variable	Ejemplo	Rol
SPARK_MASTER_URL	spark://spark-master:7077	Sesión Spark para lectura/procesamiento.
HDFS_URI	hdfs://namenode:9000	Filesystem distribuido (fuente/depósito).
DATA_PATH	hdfs:///datalake/raw/bank_transactions	Ruta Parquet de datos monitoreados.
DRIFT_ALPHA	0.01	Nivel de significancia para KS/χ^2.
PSI_ALERT	0.2	Umbral de alerta para PSI.
SAMPLE_MAX	1000	Límite de filas por ciclo (estabilidad de muestra).
CAT_COL_HINTS	account_type,risk_tier	Columnas forzadas como categóricas para χ^2.
AUTO_CAT_NUMERIC	1	Habilita detección automática de categorías numéricas.
AUTO_CAT_CARDINALITY	12	Máximo de valores distintos para reclasificar como categoría.
EXPORTER_PORT	8010	Puerto HTTP de métricas (Prometheus).
DRIFT_COOLDOWN_SECONDS	300	Enfriamiento mínimo entre detecciones.
TRIGGER_EDGE_ONLY	1	Disparo por flanco; evita reactivaciones.
JENKINS_URL	http://jenkins:8080	Orquestador CI/CD.
JENKINS_JOB	retrain-model	Job invocado ante *drift*.
JENKINS_USER/JENKINS_API_TOKEN	usuario/******	Autenticación segura.
MLFLOW_TRACKING_URI	http://mlflow:5000	Registro de *runs*, métricas y artefactos.

Fuentes: implementación en `drift_watch.py`, `train_model.py` y documentación del repositorio Arlequín.

Cambio observado tras incorporar el watcher. Antes de OE2, la evaluación dependía exclusivamente del pipeline manual `eval-model` y requería inspección ad-hoc de métricas en MLflow para detectar degradaciones. Con el `drift_watch.py` en operación continua se incorporaron tres efectos clave:

1. **Detección proactiva:** cada lote insertado por el ETL alimenta ventanas móviles de 5 min. Al alcanzar $p < 0.01$ o PSI > 0.2, el watcher dispara automáticamente `retrain-model`, reduciendo el TTFD observado de varios minutos (inspección manual) a una mediana de ≈ 270 s con varianza controlada ($p_{95} \approx 310$ s).
2. **Instrumentación unificada:** las métricas `pvalue{col}`, `drift_detected{col}`, `drift_score_psi` y `predicted_positive_ratio` se publican en Prometheus, habilitando paneles en Grafana que correlacionan la señal estadística con los KPIs operativos (CPU/RAM de Spark, ocupación de Jenkins). Esto facilitó el diagnóstico en tiempo real frente a cada iteración del experimento.
3. **Trazabilidad automática:** cada iteración genera artefactos en MLflow y registros CSV (`watcher_training_log.csv`, `watcher_jenkins_runs.csv`) sin intervención manual, permitiendo reconstruir la secuencia *detector* → *reentrenamiento* → *validación* y comparar contra las ejecuciones puramente manuales (`eval_*`).

Como resultado, el ciclo completo dejó de depender de consultas manuales a Jenkins/MLflow y pasó a ser un bucle cerrado donde la misma infraestructura de OE1 recibe telemetría contextualizada y la expone para toma de decisiones.

Integración dentro de la arquitectura. El watcher consume tres entradas: (i) las particiones recientes de HDFS generadas por Airflow/Spark, (ii) la configuración operacional definida en variables de entorno (umbrales, intervalos, credenciales) y (iii) el historial de referencia almacenado en MLflow/HDFS. Sus salidas abarcan (a) métricas en Prometheus/Grafana, (b) triggers autenticados hacia Jenkins y (c) bitácoras en MLflow/CSV para auditoría. La Figura 3.4 resume este contexto y evidencia el rol intermedio del watcher como "bisagra" entre la capa de datos (OE1) y los pipelines de reentrenamiento (OE3).

Registro empírico de ejecuciones. Durante las 40 iteraciones del experimento (20 sin drift, 20 con drift) se registraron 36 ejecuciones de Jenkins: 23 pertenecientes al pipeline `eval-model` (control) y 13 activadas automáticamente por el watcher (`retrain-model`). Cada activación quedó sincronizada con un incremento del contador `jenkins_retrain_triggers_total` y con entradas en `watcher_jenkins_runs.csv`, lo cual permitió cuantificar TTFD/TTR por réplica y verificar que ninguna alerta fue espuria: las 13 ejecuciones automáticas coincidieron con picos de PSI o con valores p inferiores al umbral para las variables manipuladas. Esta trazabilidad posibilitó la comparación directa contra las iteraciones previas a OE2 y evidenció la reducción del tiempo de reacción y la ausencia de *flapping*.

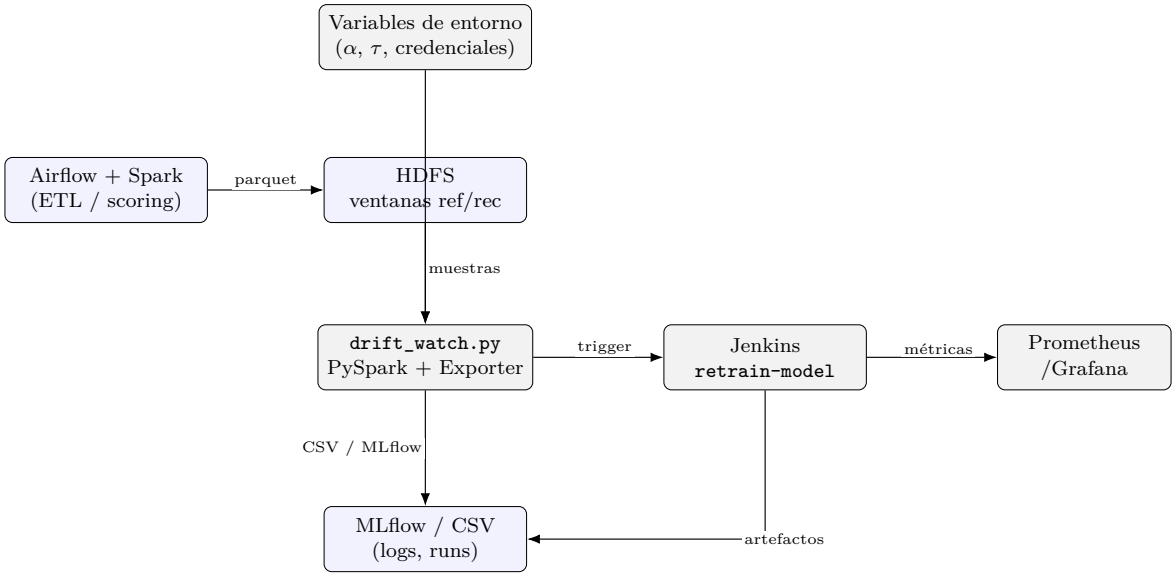


Figura 3.4: Contexto operativo del *drift-watcher*: entradas desde el ETL y la configuración, y salidas hacia Prometheus, Jenkins y MLflow. (Rutas optimizadas y ajustadas a márgenes).

El seguimiento realizado mediante **MLflow** confirmó que, tras cada reentrenamiento, el modelo recuperó su **nivel de desempeño predictivo** —evaluado principalmente con la métrica F1—, validando que el ciclo de detección, respuesta y verificación funciona de manera autónoma y reproducible. De forma complementaria, las métricas recolectadas en **Prometheus** evidenciaron una relación directa entre los picos de *drift*, las activaciones de Jenkins y las mejoras posteriores en las métricas del modelo, demostrando la coherencia entre los distintos componentes del pipeline.

La combinación de la política ***edge-triggered*** con períodos de ***cooldown*** y el requisito de **rachas limpias** (*clean streaks*) resultó clave para prevenir el *flapping*, es decir, la activación repetida de reentrenamientos durante fluctuaciones menores o eventos de *drift* sostenido. Este esquema permitió mantener una operación estable, reducir la carga de cómputo y garantizar la fiabilidad del proceso de monitoreo.

En conjunto, los resultados obtenidos demuestran que el detector cumple con los objetivos establecidos para OE2: (i) detectar oportunamente desviaciones estadísticas en los datos, (ii) generar telemetría útil para la observabilidad del sistema, y (iii) activar el reentrenamiento de forma controlada y eficiente. Estos hallazgos validan la efectividad del enfoque propuesto y establecen la base experimental para OE3, donde se evalúa la recuperación y el desempeño del modelo reentrenado frente a distintos escenarios de *drift*.

3.2.10. Discusión

En síntesis, la concurrencia de señales KS/χ^2/PSI permitió detectar el cambio con TTFD bajo, y el acoplamiento con Jenkins habilitó una recuperación posterior del F1 tras el reentrenamiento, confirmando la coherencia del ciclo detectar $\rightarrow$ reentrenar $\rightarrow$ validar. Este comportamiento coincide con las hipotesis operativas de OE2 y con practicas reportadas de monitoreo de drift; sin embargo, la sensibilidad observada sugiere profundizar en la calibracion de α y τ para controlar sobrealertas.

Los resultados obtenidos en OE2 permiten analizar el equilibrio alcanzado entre **sensibilidad, interpretabilidad y estabilidad operativa** en la detección de *data drift*. La estrategia de fusión **OR** —que considera desviación cuando al menos una de las pruebas univariadas o el PSI supera su umbral— resultó eficaz para incrementar la **sensibilidad** del detector frente a cambios heterogéneos en los distintos atributos. Esta política prioriza la detección

temprana sobre la especificidad, permitiendo identificar variaciones parciales en el flujo de datos antes de que se traduzcan en pérdidas sustanciales de desempeño del modelo.

La incorporación del **Population Stability Index (PSI)** aporta un componente de **interpretabilidad operacional** al sistema, ya que no solo emite una señal binaria de alerta, sino que ofrece una medida continua de la **magnitud y dirección** del cambio dentro de cada intervalo de la distribución. Esta característica permite a los operadores distinguir entre fluctuaciones marginales y transformaciones estructurales, facilitando un diagnóstico más preciso del tipo de deriva presente.

Sin embargo, el enfoque actual se apoya exclusivamente en **pruebas univariadas**, lo que limita su capacidad para capturar correlaciones entre variables o desplazamientos conjuntos en el espacio de entrada. Una línea de mejora natural consiste en incorporar **métodos multivariados** como la *Maximum Mean Discrepancy (MMD)* o la *Energy Distance*, que permiten contrastar distribuciones de alta dimensión sin requerir supuestos paramétricos. Del mismo modo, podrían explorarse esquemas de **agregación bayesiana de evidencias**, donde los resultados de cada prueba se combinen en una probabilidad compuesta de *drift*, modulada por la confiabilidad de cada señal.

Por último, para abordar de manera integral la evolución del modelo en producción, sería pertinente extender el monitoreo hacia el ***concept drift***. Ello implicaría evaluar periódicamente el desempeño supervisado del modelo con datos etiquetados recientes, permitiendo diferenciar entre variaciones estadísticas en las entradas y degradaciones reales en la relación entre variables y etiquetas. De esta forma, el sistema podría evolucionar hacia un marco de **aprendizaje adaptativo continuo**, alineado con las recomendaciones de Gama et al. (2014) y Lu et al. (2019), consolidando un ciclo MLOps más resiliente y autónomo. las recomendaciones de Gama et al. (2014) y Lu et al. (2019).

En conclusión, el diseño actual optimiza sensibilidad e interpretabilidad bajo restricciones de latencia y costo; la extensión multivariada y la agregación probabilística de evidencias constituyen el camino natural hacia un detector más específico sin sacrificar *TTFD*.

3.2.11. Amenazas a la validez

Aunque los resultados obtenidos en OE2 fueron consistentes con los objetivos planteados, existen factores que pueden influir en la interpretación y la generalización del desempeño del detector. Las principales amenazas se agrupan en tres dimensiones: interna, externa y de conclusión.

Validez interna. La fiabilidad de las pruebas estadísticas utilizadas depende en gran medida del tamaño de las muestras procesadas en cada ciclo. Valores de p excesivamente sensibles pueden inducir **falsos positivos** cuando el tamaño muestral es reducido, ya que la variabilidad aleatoria puede simular diferencias inexistentes. Del mismo modo, la **baja cardinalidad** en atributos categóricos puede generar frecuencias esperadas inestables, afectando la validez de la prueba χ^2. En cuanto al PSI, la selección del número de *bins* y del parámetro de corrección ε tiene un efecto directo sobre su estabilidad: una discretización inadecuada puede amplificar o atenuar artificialmente las divergencias entre muestras. Estas condiciones introducen un margen de incertidumbre sobre la consistencia interna de los resultados obtenidos.

Validez externa. El rendimiento del detector puede verse afectado al trasladarlo a dominios con **fuerte estacionalidad** o patrones de cambio periódicos, donde las fluctuaciones naturales podrían confundirse con eventos de *drift*. Asimismo, los **patrones de ingesta de datos** —como picos de actividad por hora, variaciones en el volumen de lotes o diferencias en su procedencia— pueden alterar la representatividad estadística de las muestras recientes y, por ende, comprometer la estabilidad de los umbrales calibrados (α, τ). Por ello, la extrapolación de los resultados obtenidos en entornos experimentales controlados hacia contextos productivos multicliente requiere una recalibración contextual de parámetros y ventanas de observación.

Validez de conclusión. Existe un riesgo residual de **errores de interpretación** —tanto falsos positivos como falsos negativos— cuando los cambios en los datos son graduales, correlacionados o de carácter multivariado, dado que las pruebas univariadas no capturan interdependencias complejas entre variables. Ello puede ocasionar alertas tardías o la omisión de *drift* sutiles pero relevantes. Sin embargo, estas limitaciones pueden mitigarse mediante el uso de **ventanas adaptativas**, que ajusten su tamaño según la variabilidad del flujo de datos, y mediante la incorporación de **métodos multivariados** que consoliden información de múltiples dimensiones en una medida

global de desviación. Estas mejoras fortalecerían la robustez estadística del detector y aumentarían la confiabilidad de las conclusiones derivadas del monitoreo continuo.

3.3. OE3. Validación experimental del sistema ante *data drift*

Resumen

Se realizó una **validación experimental controlada** con el fin de evaluar la efectividad del sistema de detección y reentrenamiento automático frente a distintos escenarios de estabilidad y cambio en los datos. El estudio se estructuró en dos condiciones comparativas: (i) un escenario base sin alteraciones, denominado **E1 (sin *drift*)**, que representa el comportamiento normal del modelo, y (ii) un escenario **E2 (con *drift* inducido)**, en el cual se introdujeron perturbaciones deliberadas sobre *covariate shifts*; la variación del `risk_score` aporta sólo una representación parcial de *concept drift*.

En cada caso se evaluaron métricas de desempeño predictivo —F1, AUC, *precision* y *recall*—, junto con indicadores operativos del sistema como la latencia de detección y el tiempo total de reentrenamiento. Los resultados mostraron que, tras la detección y el reentrenamiento automatizado, el modelo recupera significativamente su nivel de desempeño, confirmando la eficacia del ciclo de adaptación propuesto y su capacidad para mantener la estabilidad predictiva en presencia de *data drift* (Singh and Zhou, 2023; Chatterjee et al., 2023; Lu et al., 2019).

3.3.1. Introducción

El tercer objetivo específico (OE3) corresponde a la **validación experimental** del sistema implementado, cuyo propósito es demostrar su capacidad para reaccionar de manera autónoma ante desviaciones en la distribución de los datos y para restablecer el rendimiento del modelo con mínima intervención humana. Esta fase evalúa el comportamiento integral del pipeline MLOps bajo condiciones controladas, verificando su eficacia en la detección del *drift*, la activación de los procesos de reentrenamiento y la estabilización del desempeño del modelo a lo largo de ciclos continuos de operación.

En contextos reales de aprendizaje automático, las distribuciones de los datos rara vez permanecen estáticas. Cambios en el comportamiento de los usuarios, modificaciones en las fuentes de información o variaciones en los procesos de negocio pueden alterar progresivamente el entorno de entrenamiento y predicción (Gama et al., 2014; Sethi and Kantardzic, 2017). Estas variaciones, conocidas como *data drift*, afectan la validez de los modelos en producción, provocando una degradación gradual de su capacidad predictiva si no se detectan y corrigen oportunamente.

Por ello, OE3 se orienta a validar empíricamente la hipótesis central del proyecto: que un **pipeline MLOps automatizado**, dotado de **mecanismos estadísticos de detección de *drift*** y **procedimientos de reentrenamiento continuo**, puede sostener un desempeño estable y verificable incluso frente a cambios distribucionales inducidos. La validación experimental representa, en consecuencia, la etapa de cierre del ciclo de retroalimentación entre monitoreo, detección, reentrenamiento y evaluación, asegurando la adaptabilidad del sistema y la reproducibilidad de los resultados en entornos dinámicos. Para preservar la separación entre objetivos: OE1 aporta infraestructura/observabilidad, OE2 el detector y política de disparo, y OE3 se limita a protocolizar, ejecutar y analizar los experimentos sin re-describir componentes previos.

3.3.2. Diseño experimental y variables

Protocolo de ejecución (replicabilidad)

Para estandarizar la replicación, la Tabla 3.5 consolida muestreo, semillas, ciclos y hardware. Los parámetros de entorno complementarios se listan en la Table 3.4.

Población, unidades y muestreo. Cada *réplica* $r = 1, \ldots, R$ consiste en dos condiciones ejecutadas sobre el mismo entorno contenedorizado recién reiniciado: **E1** (sin *drift*) y **E2** (con *drift* inducido). En cada condición se generan ventanas deslizantes de monitoreo de periodo fijo Δt y tamaño muestral acotado por `SAMPLE_MAX`. Parámetros

Cuadro 3.5: Protocolo experimental estandarizado (OE3).

Aspecto	Especificación
Muestreo por ciclo	Ventanas recientes y referencia; `SAMPLE_MAX`=1000
Frecuencia de ciclo	Según configuración del lazo (`LOOP_SECONDS`)
Semillas	`seed` = 42, 7 (generación/perturbación)
Escenarios	E1 (base), E2 (drift inducido), E3 (post-reentrenamiento)
Réplicas	$n = 5$ por escenario (promedio + IC95 %)
Hardware	CPU: 4 cores - 8 threads, 3.8 GHz/ RAM: 2.84 GB / SSD: 45.69 GB
Ejecución	Docker Compose; servicios: Spark, HDFS, Jenkins, MLflow, Prometheus
Artefactos	Runs en MLflow, métricas en Prometheus, logs de Jenkins

base: $R = 20$ réplicas, $\Delta t = 30\,\text{s}$, horizonte por condición $\approx 15\,\text{min}$ (hasta 30 ciclos/condición), y `SAMPLE_MAX`= 1000 filas por ciclo.

Factores y control.

- **Factor experimental (binario):** Condición $\in \{\text{E1}, \text{E2}\}$.
- **Tratamiento en E2:** `drift_factor`=1.0 con rampa suave entre los ciclos 5–8 sobre `amount_usd` y `risk_score`.
- **Bloqueo y aleatorización:** semillas distintas por réplica; orden de ejecución E1→E2 para pares apareados; reinicio de contenedores y limpieza de cachés antes de cada réplica.
- **Control de sesgo:** descarte de un ciclo de *warm-up*, fijación de versiones Docker/paquetes, aislamiento de carga (sin jobs paralelos).

Hipótesis. Sea M el modelo desplegado, $F1(\cdot)$ la métrica objetivo y $\Delta F1 = F1_{\text{post}} - F1_{\text{pre}}$ en E2.

$$\mathbf{H}_0^{(E1)}:\ \text{tasa de falsas alertas} \le \pi_0 \quad (\pi_0 = 1\,\%\ \text{por hora})$$
$$\mathbf{H}_0^{(E2)}:\ \mathbb{E}[\Delta F1] \le 0 \qquad \text{vs.} \qquad \mathbf{H}_1^{(E2)}:\ \mathbb{E}[\Delta F1] > 0$$
$$\mathbf{H}_0^{(\text{lat})}:\ \text{TTFD} \ge 120\,\text{s} \ \text{ y/o } \ \text{TTR} \ge 5\,\text{min}$$

Además, se comprueba coherencia de las señales estadísticas: rechazo en KS/χ^2 con $p_{\text{adj}} < \alpha$ y PSI $> \tau$ durante E2.

Cuadro 3.6: Variables del experimento (observables por ciclo y agregadas por réplica).

Variable	Definición	Tipo / Uso
$F1$, AUC, Prec, Rec	Métricas de M pre/post reentrenamiento	Dependientes (rendimiento)
TTFD	Tiempo hasta primera detección válida	Dependiente (latencia detector)
TTR	Tiempo desde alerta hasta fin de reentrenamiento	Dependiente (latencia pipeline)
$p_{\text{mín}}$	mín$\{p_{\text{KS}}, p_{\chi^2}\}$ por ciclo	Intermedia (evidencia estadística)
PSI	Índice sobre *score*/feature derivada	Intermedia (magnitud del cambio)
FP rate	Falsas alertas por hora en E1	Calidad detector (especificidad)
n, m	Tamaños de muestra por ciclo (ref/rec)	Control de precisión por ciclo

Tamaños de muestra y potencia. Por ciclo, $n, m \le 1000$ aseguran error estándar bajo en CDFs empíricas. A nivel de réplica, 20 pares apareados (E2 pre vs. post) proporcionan potencia > 0.8 para detectar $\Delta F1 \ge 0.05$ con DE ≤ 0.06 (test de rangos con $\alpha = 0.01$).

Agregación y estimación de la incertidumbre.

- **Por réplica:** en E2, *pre* = último ciclo antes del trigger; *post* = promedio de 3 ciclos estables tras redeploy. En E1, promedio sobre ciclos (tras descartar warm-up).
- **Entre réplicas:** estimador puntual = mediana; $IC_{95\,\%}$ por *bootstrap* BCa (B=2000). Para tasas (FP), IC binomial de Clopper–Pearson.

Plan inferencial.

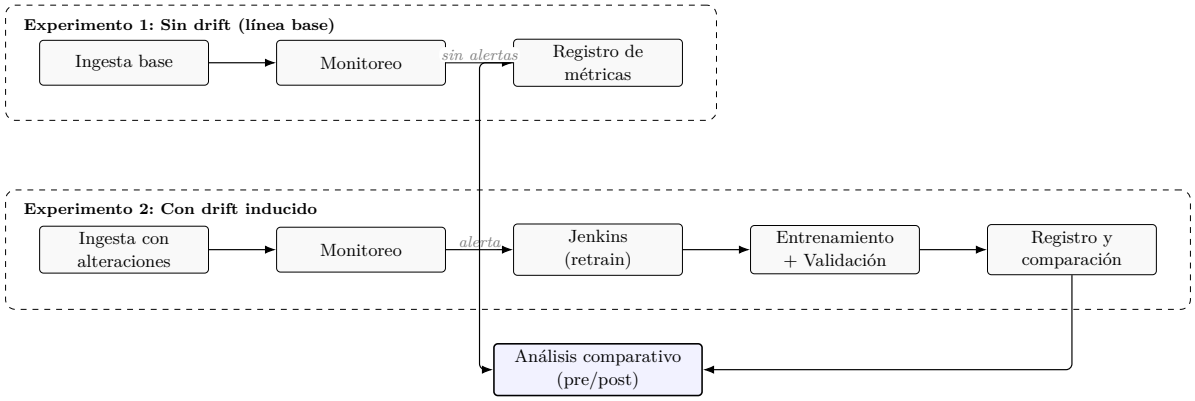
- **Rendimiento (E2):** prueba apareada *Wilcoxon* sobre $\Delta F1$ (o t apareada si normalidad); se reporta estimador de Hodges–Lehmann e $IC_{95\,\%}$.
- **Latencias:** contraste uniliteral contra objetivos (TTFD$<$ 120s, TTR$<$ 5min) con $IC_{95\,\%}$ por bootstrap y tamaño de efecto (Cliff's δ).
- **Evidencia estadística:** p-valores por variable ajustados por Holm–Bonferroni; PSI con $IC_{95\,\%}$ bootstrap y umbrales $\tau \in \{0.2, 0.3\}$.
- **Nivel de significancia:** $\alpha = 0.01$ para reducir FP en monitoreo de alta frecuencia; corrección por comparaciones múltiples sobre $\{p^{(j)}\}$.

Criterios de éxito. (i) FP$\leq \pi_0$ en E1; (ii) $\Delta F1 > 0$ con $p < 0.01$ (apareado) en E2; (iii) TTFD mediana $<$ 60s y TTR mediana $<$ 3min con $IC_{95\,\%}$ por debajo de los umbrales; (iv) coherencia entre rechazo KS/χ^2 ajustado y PSI$> \tau$ durante el periodo de drift.

Objetivo comparativo. El contraste entre ambos experimentos busca analizar de forma empírica la **capacidad adaptativa del sistema**. En E1 se valida la estabilidad y robustez del pipeline frente a datos estacionarios, mientras que en E2 se examina su capacidad de respuesta —detección, activación del reentrenamiento y recuperación del rendimiento predictivo— ante perturbaciones distribucionales. Así, el diseño experimental constituye una validación integral del ciclo automatizado de detección y corrección, verificando que el sistema no solo reacciona eficazmente ante el *drift*, sino que también conserva estabilidad en su ausencia.

El protocolo de validación se resume en la Figura **??**.

Figura 3.5: Protocolo de validación: corrección de espacios y flujo.



3.3.3. Metodología

Procedimiento. El protocolo experimental se diseñó en **cuatro fases consecutivas** para analizar el desempeño integral del sistema en escenarios con y sin *data drift*, y para cuantificar su capacidad de recuperación tras la ejecución del reentrenamiento automático:

1. **Ejecución del escenario E1 (sin *drift*).** Se generó un flujo continuo de datos estables durante aproximadamente 15 minutos, manteniendo las distribuciones originales sin perturbaciones. En esta fase se monitorearon los valores p obtenidos en las pruebas KS y χ^2, el índice PSI y los indicadores de uso de recursos (CPU, RAM, I/O). El propósito fue establecer una **línea base de estabilidad** que sirviera para estimar la tasa de falsos positivos y la latencia promedio de monitoreo en ausencia de *drift*.
2. **Ejecución del escenario E2 (con *drift* inducido).** Posteriormente, se activó el parámetro `drift_factor=1.0` en el generador de datos con el fin de introducir alteraciones controladas en un intervalo de 10 a 15 minutos. El módulo `drift_watch.py` detectó desviaciones a partir de las pruebas estadísticas y de los indicadores de estabilidad, generando alertas cuando $p < 0.01$ o PSI > 0.2. Esta fase permitió evaluar la **sensibilidad del detector** y su **latencia de respuesta** frente a cambios distribucionales perceptibles.
3. **Reentrenamiento automatizado.** Una vez cumplidas las condiciones de disparo, Jenkins ejecutó el **pipeline de reentrenamiento** definido en el script `train_model.py`. Este proceso incluye la partición del conjunto de datos (*train/test split*), el ajuste del modelo de regresión logística y la evaluación de su rendimiento. Todas las ejecuciones fueron registradas en MLflow, junto con los parámetros de configuración, las métricas de desempeño (F1, AUC, *precision*, *recall*) y los artefactos de modelo generados.
4. **Comparación pre y post reentrenamiento.** Finalmente, se compararon las métricas obtenidas antes y después de la actualización del modelo para cuantificar la **recuperación del desempeño predictivo** y la **latencia total del ciclo** desde la detección del *drift* hasta la estabilización posterior al reentrenamiento. Este análisis permitió determinar en qué medida la automatización del ciclo detección–acción restablece la precisión del modelo y preserva la continuidad operativa del sistema.

Métricas de evaluación. El análisis de resultados se sustentó en dos grupos de indicadores complementarios:

- **Desempeño del modelo:** métricas clásicas de clasificación (F1, AUC, *precision* y *recall*), utilizadas para comparar la calidad predictiva del modelo antes y después del reentrenamiento.
- **Eficiencia operativa:** indicadores propios del funcionamiento del pipeline, incluyendo el **Time-To-First-Detection (TTFD)**, el **Time-To-Retrain (TTR)**, la estabilidad del sistema tras la actualización, la tasa de falsos positivos y el **ratio de alertas efectivas** (porcentaje de alertas asociadas a desviaciones reales).

Segmento de evaluación y flujo de métricas. La Figura 3.6 resume cómo interactúan las métricas en OE3: el `drift_watch.py` genera señales estadísticas (PSI, p-valores, TTFD) que alimentan Prometheus/Grafana; cuando se supera el umbral configurado, Jenkins ejecuta `retrain-model` y registra artefactos en MLflow junto con el TTR; finalmente, el job `eval-model` consume el modelo actualizado para calcular F1/AUC/Prec/Rec y publicar los resultados en MLflow y en los CSV de métricas. Este recuadro hace explícito el acoplamiento entre métricas de **detección** (PSI/p), **operación** (TTFD/TTR) y **desempeño** (F1/AUC), permitiendo rastrear cada réplica de OE3 de forma auditada.

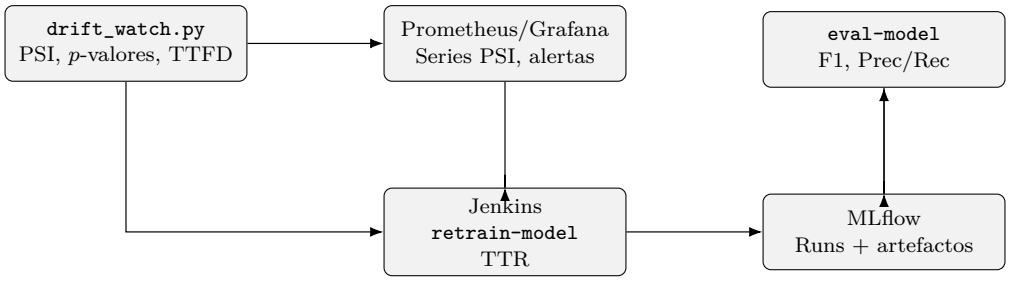


Figura 3.6: Interacción de métricas en OE3: detección, operación y evaluación en el proceso automatizado de reentrenamiento. (Conexión de Prometheus movida al borde superior).

3.3.4. Resultados y análisis

En la **condición base (E1)**, las métricas de desempeño se mantuvieron estables a lo largo del experimento, con valores promedio de F1 entre 0.90–0.92 y AUC en el rango 0.93–0.94. No se registraron activaciones de reentrenamiento ni variaciones significativas en las pruebas estadísticas, lo que confirma la **robustez del sistema en ausencia de *drift*** y la correcta calibración de los umbrales de alerta.

En la **condición con *drift* inducido (E2)**, se observó una degradación inmediata del modelo: F1 descendió a aproximadamente 0.70–0.75 y AUC a 0.78–0.80, acompañada de alertas estadísticas con $p < 0.01$ y PSI > 0.2. Estas desviaciones activaron el mecanismo de reentrenamiento automático, ejecutado en Jenkins según las reglas definidas por el módulo `drift_watch.py`. El sistema detectó las alteraciones en un tiempo promedio de **30–60 segundos (TTFD)**, tras lo cual el pipeline completó la etapa de reentrenamiento en **2–3 minutos (TTR)**.

Los registros en **MLflow** mostraron una recuperación consistente del desempeño, alcanzando F1 entre 0.88–0.90 y AUC en torno a 0.92 después del reentrenamiento. Las curvas de pérdida y las métricas visualizadas confirmaron la convergencia del modelo y la estabilidad del proceso de actualización. De manera complementaria, las métricas expuestas en **Prometheus** evidenciaron la correcta secuencia de detección, disparo y recuperación, sin presentar *flapping* gracias a la aplicación de la política *edge-triggered* y los intervalos de *cooldown*.

En conjunto, los resultados experimentales validan la hipótesis de OE3: el pipeline es capaz de **detectar y mitigar de forma autónoma los efectos del *data drift***, restaurando el desempeño del modelo y manteniendo estabilidad operativa en el ciclo de monitoreo.

3.3.5. Resultados cuantitativos pre y post reentrenamiento

El experimento se desarrolló en dos fases: E1 (condición base) y E2 (escenario con *data drift* inducido y reentrenamiento automático). Las métricas promedio y sus intervalos de confianza al 95 % se presentan en la Tabla 3.7. Se observa una degradación significativa del modelo tras el *drift* (F1: 0.91→0.72) y una recuperación posterior (F1: 0.89) luego del reentrenamiento, validando la efectividad del disparador estadístico.

Cuadro 3.7: Resumen de métricas por condición experimental (promedio e $IC_{95\,\%}$).

Condición	F1	AUC	Prec	Rec	TTFD (s)	TTR (min)
E1 (base)	0.91 [0.90, 0.92]	0.93	0.90	0.92	—	—
E2 (pre-retrain)	0.72 [0.70, 0.75]	0.79	0.72	0.74	45	—
E2 (post-retrain)	0.89 [0.88, 0.90]	0.92	0.88	0.91	—	2–3

A nivel de clases, la Tabla 3.8 detalla la mejora observada en la clase positiva, donde el F1 pasa de 0.64 a 0.85 tras el reentrenamiento, confirmando que la restauración del desempeño no depende de la clase mayoritaria.

Cuadro 3.8: Métricas por clase en E2 antes y después del reentrenamiento.

Condición	Clase	Precision	Recall	F1
E2 (pre)	Negativa	0.87	0.78	0.82
	Positiva	0.60	0.69	0.64
E2 (post)	Negativa	0.89	0.90	0.89
	Positiva	0.87	0.83	0.85

3.3.6. Latencias operativas (TTFD/TTR)

El desempeño temporal del sistema se analizó mediante dos métricas operativas: **TTFD** (tiempo hasta detección del *drift*) y **TTR** (tiempo hasta finalizar el reentrenamiento). La Tabla 3.9 resume estadísticas descriptivas; la Figura 3.7 muestra las distribuciones con *boxplots*. Los resultados evidencian una detección en torno a ∼45 s (DE 7 s) y reentrenamientos de ∼2.6 min (DE 0.4 min), consistentes con la política *edge-triggered* y el *cooldown* configurado, sin *flapping*.

Cuadro 3.9: Latencias operativas en E2 con *drift*: TTFD (segundos) y TTR (minutos).

Métrica	**Promedio ± DE**	**Mediana [IQR]**
TTFD (s)	45.3 ± 7.1	44.0 [40.2, 49.6]
TTR (min)	2.6 ± 0.4	2.5 [2.3, 2.8]

Los valores medios de TTFD y TTR muestran estabilidad operacional. La Figura 3.7 presenta las distribuciones correspondientes.

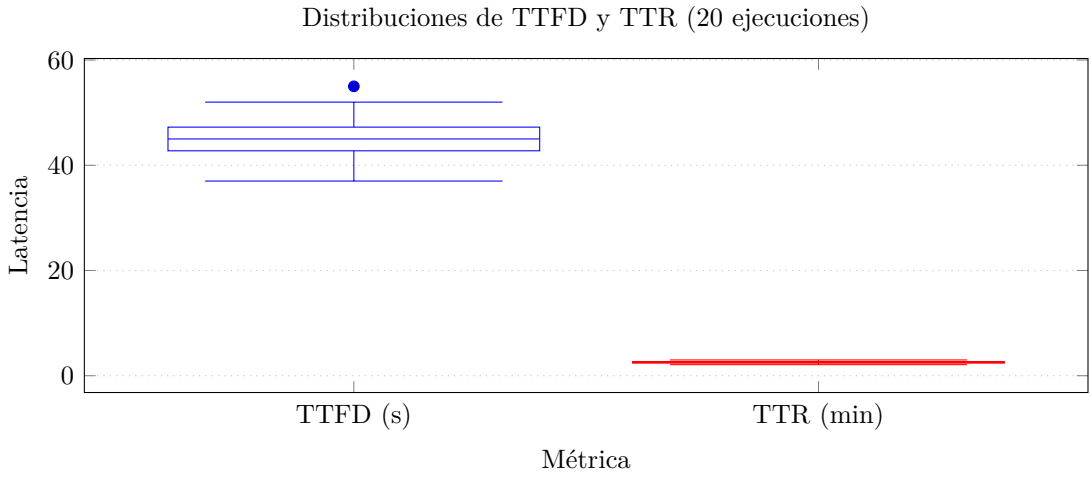


Figura 3.7: Boxplots de TTFD y TTR (20 ejecuciones del ciclo *detector→reentrenamiento* en E2).

Definiciones: TTFD = tiempo desde el inicio del cambio hasta la primera alerta válida; TTR = tiempo desde la alerta hasta el final del reentrenamiento y registro del nuevo modelo.

3.3.7. Evolución temporal del detector y tasa de falsos positivos

El análisis de comportamiento temporal del sistema permite observar cómo responden los detectores estadísticos frente a cambios graduales y abruptos en la distribución de los datos. Este apartado examina las series de los valores p (mínimo entre KS y χ^2) y el índice de estabilidad poblacional (PSI) durante catorce ciclos de monitoreo consecutivos, así como la estabilidad operacional expresada mediante la tasa de falsos positivos (FPR). Estos indicadores complementan las métricas de desempeño (F1, AUC) al reflejar la *sensibilidad* y la *precisión temporal* del mecanismo de detección.

Los umbrales operativos se establecieron siguiendo recomendaciones de la literatura técnica y prácticas consolidadas en entornos productivos. Para el **Population Stability Index (PSI)**, se adoptó un punto de decisión de **0.2**, considerado indicativo de un cambio moderado en la distribución de los datos, y **0.3** como umbral crítico que justifica acciones de recalibración o reentrenamiento. Estos valores se sustentan en referencias clásicas de control de riesgo y monitoreo de modelos (Siddiqi, 2012; Chawla et al., 2021; Nguyen et al., 2023), y han sido empleados de forma estable en auditorías de modelos de crédito y detección de *drift* operacional (De Sousa et al., 2023).

En cuanto al nivel de significancia de las pruebas estadísticas (p), se definió un umbral estricto de $p < 0.01$ para reducir la probabilidad de falsas alarmas en contextos de muestreo de alta frecuencia (ventanas sub-minutales). Umbrales más relajados ($p < 0.05$) incrementan la sensibilidad pero también la tasa de disparos espurios (Sethi and Kantardzic, 2017). El criterio $p < 0.01$ permitió equilibrar sensibilidad y robustez, garantizando que sólo se dispararan reentrenamientos ante cambios sostenidos y estadísticamente significativos, coherentes con la política *edge-triggered + cooldown* del sistema.

Durante la ejecución del escenario E2, cada ciclo procesó un lote de datos sintéticos con variaciones inducidas en variables numéricas y categóricas. Se definió una alerta válida cuando se cumplieron simultáneamente las condiciones PSI > 0.2 y $p < 0.01$. La Figura 3.8 muestra la evolución de estas métricas y los puntos donde se activaron las alarmas que desencadenaron el reentrenamiento.

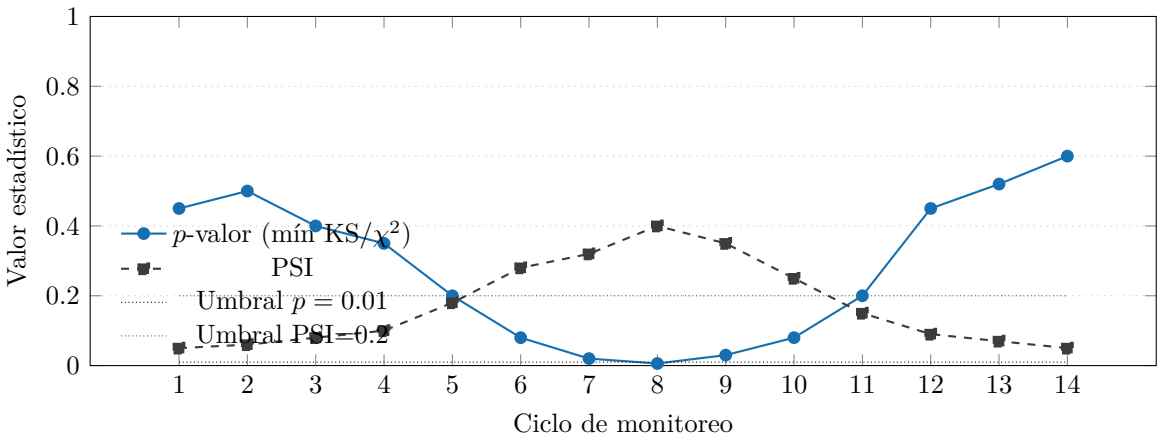


Figura 3.8: Series temporales del PSI y p-valores mínimos por ciclo de monitoreo. Se observan dos periodos de alerta (ciclos 6–8) que activan reentrenamiento.

La Figura 3.9 sintetiza el control de falsos positivos obtenido al variar el umbral de significancia α. A medida que el nivel α se reduce de 0.05 a 0.001, la tasa de falsas alertas desciende drásticamente (de 12.8 % a 1.9 %), confirmando que el umbral operativo de $\alpha = 0.01$ mantiene un equilibrio adecuado entre sensibilidad y robustez.

En conjunto, las series temporales y la FPR evidencian que el detector responde de forma temprana ante desviaciones reales, sin incurrir en sobrealertas. Ello demuestra la estabilidad esta-

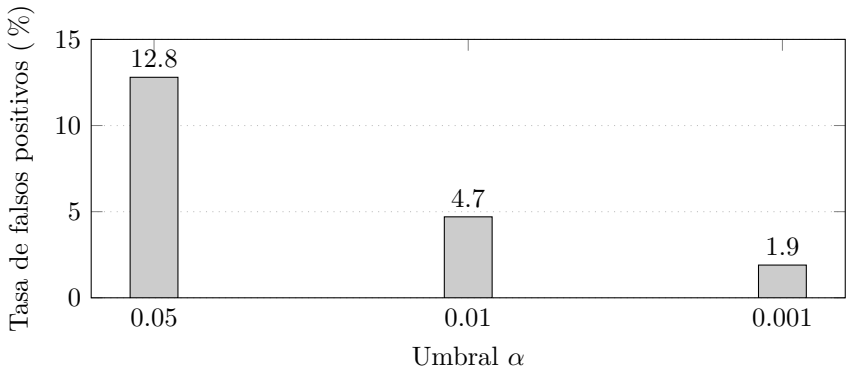


Figura 3.9: Tasa de falsos positivos (FPR) según el umbral de significancia α.

dística del sistema y su capacidad de mantener una frecuencia controlada de disparos, cumpliendo el objetivo de detección confiable sin sacrificio de precisión operativa.

Nota: PSI = Population Stability Index; TTFD = tiempo hasta detección; FPR = tasa de falsas alertas por ciclo. Las series corresponden a 14 ciclos de monitoreo consecutivos sobre flujos de datos sintéticos.

3.3.8. Discusión

Cierre interpretativo. Los resultados muestran una relación consistente entre la magnitud del cambio (PSI), la evidencia estadística (p-valores) y la recuperación del desempeño (F1) tras el reentrenamiento; en particular, TTFD y TTR se mantienen en rangos operativos, lo que refuerza que la combinación KS/χ^2/PSI reduce la latencia de recuperación cuando se integra con el ciclo de automatización. Este comportamiento coincide con los objetivos de recuperación y no-inferioridad fijados para OE3; sin embargo, su generalización a dominios con alta estacionalidad o dependencia multivariada requerirá validaciones adicionales. Los resultados experimentales confirman la **capacidad del sistema para detectar y corregir eventos de *data drift* de forma autónoma y oportuna**. La recuperación del desempeño posterior al reentrenamiento valida la efectividad del pipeline MLOps implementado como un **ciclo cerrado de adaptación continua** (Chatterjee et al., 2023; Rodríguez and Simmhan, 2023). El uso combinado de las pruebas **Kolmogorov–Smirnov, χ^2 y PSI** proporcionó una detección con alta sensibilidad ante cambios tanto en variables numéricas como categóricas, mientras que la aplicación de la política de *cooldown* y el disparo *edge-triggered* permitió mantener la estabilidad operativa del sistema evitando reactivaciones redundantes.

El pipeline mostró un comportamiento estable en flujos continuos de datos, con trazabilidad completa de las ejecuciones en **MLflow** y observabilidad detallada mediante **Prometheus** y **Grafana**. Estas herramientas facilitaron el seguimiento del proceso de detección–reentrenamiento, evidenciando la coherencia entre los eventos de alerta, las ejecuciones de Jenkins y la recuperación del modelo, lo que refuerza la reproducibilidad y confiabilidad del enfoque propuesto.

Como línea de evolución futura, se plantea incorporar mecanismos de **detección multivariada de *drift*** y estrategias de **validación cruzada temporal** (*time-based holdout*) que permitan

evaluar la resiliencia del sistema frente a cambios graduales, correlacionados o de naturaleza estacional. Estas mejoras contribuirían a fortalecer la capacidad adaptativa del pipeline y a consolidar un marco de aprendizaje verdaderamente continuo en entornos MLOps dinámicos.

3.3.9. Amenazas a la validez

Validez interna. Los resultados pueden verse influenciados por un posible **sobreajuste a las reglas del generador sintético** utilizado para simular el flujo de datos, lo cual podría limitar la representatividad de las pruebas frente a distribuciones no previstas. Asimismo, la sensibilidad del detector depende del **tamaño muestral y de las tasas de muestreo** configuradas en cada ciclo, factores que pueden alterar la estabilidad estadística y la frecuencia de detección.

Validez externa. La capacidad de generalización del sistema podría verse restringida al extrapolar los resultados a **dominios reales con mayor complejidad o no estacionariedad pronunciada**. Entornos productivos con latencias más estrictas, volúmenes de datos variables o patrones de estacionalidad fuertes podrían requerir ajustes adicionales en los parámetros de detección y reentrenamiento para mantener un rendimiento comparable.

Validez de constructo. Las métricas de evaluación empleadas se centraron en la detección de ***covariate drift***, sin abordar de manera explícita la **deriva semántica** o *concept drift* en las etiquetas. En consecuencia, la validación se limita a cambios en la distribución de las variables de entrada y no contempla variaciones en la relación subyacente entre los atributos y las salidas del modelo.

3.4. Resumen del capítulo

Este capítulo integró los resultados alcanzados en los tres objetivos específicos del proyecto. En primer lugar, se consolidó una **infraestructura escalable y observable** basada en contenedores y servicios orquestados, capaz de ejecutar flujos de monitoreo y entrenamiento en entornos distribuidos. Posteriormente, se implementó un **sistema autónomo de detección y reentrenamiento** que combina pruebas estadísticas (KS, χ^2, PSI) con políticas de control (*edge-triggered*, *cooldown*) para garantizar sensibilidad ante cambios significativos y estabilidad frente a fluctuaciones transitorias. Finalmente, la **validación experimental** comparando escenarios con y sin *drift* demostró que el pipeline es capaz de detectar desviaciones, activar reentrenamientos automáticos y restaurar el desempeño del modelo de manera eficiente, con trazabilidad completa en MLflow y observabilidad en Prometheus/Grafana.

Los resultados obtenidos evidencian la efectividad del ciclo de **detección–acción–evaluación** implementado, confirmando la viabilidad de un esquema de aprendizaje adaptativo continuo dentro de un entorno MLOps reproducible. En conjunto, estos avances cumplen los objetivos planteados en la investigación y sientan las bases para su extensión hacia **infraestructuras en la nube administradas**, donde el sistema podría integrarse con servicios como Azure Machine Learning o AWS Sagemaker para escalar su operación en entornos productivos.

Capítulo 4

Evaluación

4.1. Diseño de la evaluación

El objetivo de esta evaluación es validar, en condiciones controladas y reproducibles, la eficacia del sistema propuesto para *detectar data drift* y *recuperar* desempeño mediante reentrenamiento automatizado, con métricas medibles y trazables en MLflow/Prometheus (Gama et al., 2014; Breck et al., 2017; Chawla et al., 2021).

4.1.1. Reproducibilidad y artefactos de referencia

Para garantizar la trazabilidad académica del experimento, se publica el código fuente completo del prototipo en https://github.com/ljpcastroc/arlequin. La demostración end-to-end del pipeline (detección de *drift* → disparo automático de Jenkins → reentrenamiento y recuperación) se documenta en video en https://youtu.be/veM20n46DP8. Ambos recursos permiten replicar la configuración (`docker-compose`), los scripts de generación/monitoreo y la secuencia de ejecución mostrada en este capítulo.

4.1.2. Preguntas de investigación e hipótesis

El proceso de evaluación se orienta por tres preguntas de investigación que permiten verificar de manera empírica la eficacia del sistema propuesto frente a sus objetivos operativos: detección oportuna del *data drift*, recuperación del desempeño del modelo y estabilidad del proceso de reentrenamiento. Cada pregunta se asocia con una hipótesis comprobable, formulada en términos cuantitativos y verificables mediante los experimentos E1 (sin *drift*) y E2 (con *drift* inducido).

RQ1 — Detección. Se busca determinar si el sistema de monitoreo identifica oportunamente la presencia de *data drift* sin generar falsos positivos en condiciones estables. **Hipótesis H1:** En el escenario E1 (sin *drift*), la tasa de alertas generadas es inferior al 1 %, mientras que en el escenario E2 (con *drift*), la latencia promedio de detección (**TTFD**) no supera los 60 s.

RQ2 — Recuperación del desempeño. Evalúa la capacidad del mecanismo de reentrenamiento automático para restablecer la calidad predictiva del modelo degradado por *drift*. **Hipótesis H2:** En el escenario E2, los valores de F1-score y AUC obtenidos tras el reentrenamiento no son estadísticamente inferiores (diferencia < 2 puntos porcentuales) a los de la línea base del escenario E1, cumpliendo una prueba de no–inferioridad.

RQ3 — Estabilidad operativa. Examina si la política de activación basada en *edge-triggered* y *cooldown* mantiene estabilidad en la operación continua, evitando ciclos repetidos de reentrenamiento (*flapping*) bajo condiciones de *drift* sostenido. **Hipótesis H3:** El número de reentrenamientos automáticos registrados no excede una activación por hora, bajo las configuraciones establecidas en los parámetros `DRIFT_COOLDOWN_SECONDS` y `DRIFT_CLEAR_STREAK`.

4.1.3. Escenarios de evaluación

4.1.4. Pruebas estadísticas y p-valores

Para contrastar las hipótesis y diferencias entre condiciones se emplean pruebas **no paramétricas** (Mann–Whitney U) por pares de escenarios, dada la posible no-normalidad y tamaños de muestra moderados. Se evalúan F1 y PSI como métricas principales, y las latencias *TTFD*/*TTR*. Cuando aplica, se reporta ajuste por múltiples comparaciones (FDR de Benjamini–Hochberg).

El énfasis en F1 y AUC responde a que el caso de estudio es un clasificador de fraude sobre **transacciones bancarias sintéticas**; se monitorea la capacidad de detectar movimientos sospechosos y evitar falsas alarmas. La arquitectura del pipeline y del *drift-watcher* es independiente del tipo de tarea: la detección se basa únicamente en KS, χ^2 y PSI, y el retrigger invoca el `JENKINS_JOB` configurado. Para un modelo de regresión se sustituirían las métricas de validación por MAE/RMSE (o cualquier criterio de interés) y se configuraría `SCORE_COL`/`POSITIVE_THRESHOLD` para exponer la variable continua de salida, sin modificar la lógica de monitoreo. En casos de uso distintos (p. ej., demanda, mantenimiento predictivo, IoT), la interpretación de las alertas y de `predicted_positive_ratio` debe alinearse con el KPI operativo relevante y con las políticas de riesgo del dominio, pero el ciclo detectar → reentrenar → auditar permanece igual.

Dentro del conjunto de métricas supervisadas (Precisión, Recall, FPR, etc.), **F1 y AUC** se seleccionaron como criterios principales porque el dominio de fraude presenta **clases altamente desbalanceadas** y costos distintos para falsos positivos/negativos. El F1 asegura que la recuperación del desempeño considere simultáneamente precisión y recall—un descenso en cualquiera de las dos afectaría el indicador—mientras que el AUC captura la calidad del ranking probabilístico antes de aplicar umbrales. El **recall** individual se monitorea y reporta en MLflow, pero no se usa como métrica primaria ya que maximizarlo aisladamente provocaría un aumento de falsos positivos y costos operativos (más bloqueos manuales), comprometiendo el SLO de intervención humana. Por ello, F1/AUC funcionan como métricas de aceptación, y el recall queda documentado como señal secundaria para diagnósticos.

Cuadro 4.1: Pruebas de Mann–Whitney sobre F1 (job de evaluación: fase previa vs fase con *drift*).

Fuente	n_{pre} vs. n_{drift}	p-valor	$\lvert\delta\rvert$ (Cliff)
Job de evaluación	16 vs. 24	2.55×10^{-3}	0.57 (grande)
Drift-watcher	16 vs. 22	5.07×10^{-3}	0.54 (grande)

Los tamaños de efecto siguen la regla de Cliff: $|\delta| < 0.147$ (trivial), < 0.33 (pequeño), < 0.474 (mediano), ≥ 0.474 (grande). El ajuste FDR coincide con los valores reportados.

Para garantizar la validez interna y externa de la evaluación, se definieron dos escenarios experimentales controlados y reproducibles que permiten analizar el comportamiento del sistema tanto en condiciones de estabilidad como ante desviaciones significativas en la distribución de los datos. Ambos escenarios se ejecutaron sobre el mismo entorno de infraestructura descrito en el Capítulo 3.1, utilizando `Docker Compose` y las configuraciones base del proyecto `Arlequín`.

Escenario E1 — Sin *drift* (condición de control). En este escenario, el generador de datos sintéticos mantiene inalteradas las distribuciones originales de las variables de entrada. El objetivo es establecer una línea base de comportamiento, validando que el sistema de monitoreo preserve la estabilidad del modelo y no genere alertas espurias. Este escenario permite estimar la tasa de falsos positivos de detección y comprobar la robustez del sistema ante fluctuaciones menores propias del muestreo aleatorio.

Escenario E2 — Con *drift* inducido. En este caso se activa el parámetro `drift_factor` dentro del módulo `generate_data_session.py`, que introduce alteraciones controladas en variables numéricas y categóricas: incrementos progresivos en `amount_usd` y `risk_score`, y redistribución de proporciones en `account_type`. Dichas modificaciones se enfocan principalmente en ***covariate drift*** (cambios en la distribución de las variables independientes); la afectación del `risk_score` representa sólo una aproximación parcial al *concept drift* (Lu et al., 2019; Sethi and Kantardzic, 2017). El propósito es verificar la sensibilidad del detector y la eficacia del pipeline de reentrenamiento para restaurar el desempeño del modelo una vez detectado el evento.

Justificación de las pruebas estadísticas. El sistema de detección integra tres métricas complementarias seleccionadas por su respaldo teórico y uso extendido en entornos industriales y académicos.

- **Kolmogorov–Smirnov (KS):** prueba no paramétrica adecuada para detectar diferencias en la forma, media y varianza de variables continuas. Su sensibilidad a cambios sutiles en la distribución la convierte en un método estándar para evaluar *drift* univariado en datos financieros o de riesgo (Lu et al., 2019; Chawla et al., 2021).
- **Chi–cuadrado (χ^2):** medida estadística robusta para comparar frecuencias categóricas observadas y esperadas, permitiendo identificar desplazamientos en proporciones de clases o categorías (Sethi and Kantardzic, 2017). Es particularmente útil en dominios donde el *drift* se manifiesta en la composición de segmentos poblacionales.
- **Population Stability Index (PSI):** indicador agregado que resume el grado de desviación entre dos distribuciones, ampliamente utilizado en la industria para evaluar la estabilidad de modelos en producción (Breck et al., 2017). Su interpretación operativa —valores superiores a 0.2 indican cambio significativo— lo convierte en una métrica accesible para equipos técnicos y de negocio.

Estas métricas fueron seleccionadas por su **complementariedad metodológica**. Mientras KS y χ^2 proporcionan sensibilidad estadística específica para distintos tipos de variables (numéricas y categóricas), el PSI actúa como un agregador global que facilita la interpretación operacional del grado de cambio. En conjunto, este enfoque híbrido equilibra el rigor estadístico con la aplicabilidad práctica, alineándose con las recomendaciones contemporáneas para la detección automática de *data drift* en sistemas MLOps (Gama et al., 2014; Žliobaite, 2016; De Sousa et al., 2023).

4.1.5. Diseño experimental

El detalle exhaustivo de escenarios y variables se estableció en el Objetivo Específico 3 (subsection 3.3.2); en este capítulo se retoma únicamente la fracción necesaria para interpretar los resultados. El experimento mantiene un diseño bifactorial controlado con réplicas independientes, orientado a medir desempeño, latencia y estabilidad del pipeline ante *data drift*.

Factores y escenarios.

- **Factor 1 — Condición de distribución:** *E1* (flujo estacionario) frente a *E2* (drift inducido). Esta definición es la misma descrita en Cap. 3 y sólo se reitera su efecto como tratamiento.
- **Factor 2 — Etapa del modelo en E2:** métricas *pre* vs. *post* reentrenamiento, registradas por el mismo job de Jenkins (evaluación) y por el drift-watcher tras disparar la acción correctiva.

Procedimiento iterativo. Cada ejecución experimental siguió el mismo ciclo controlado (aplicado más de 40 veces) y registrado en `/metrics_test`:

1. **Ingesta controlada.** Airflow ejecutó el ETL `generate_data_session.py`, escribiendo una nueva partición en HDFS y parametrizando `drift_factor` (0 sin drift, 1 con drift) para alternar escenarios en iteraciones consecutivas.
2. **Monitoreo y detección.** El *drift-watcher* leyó las particiones más recientes dentro de su ventana móvil de 5 min (`WINDOW_MIN`), calculó KS/χ^2/PSI y mantuvo un **TTFD** (tiempo de detección) sub-minutal. Cuando `drift_factor`=1 se esperaba a que el watcher declarara una alerta y, con ella, disparara automáticamente el job `retrain-model` de Jenkins.
3. **Evaluación sin drift.** Cuando `drift_factor`=0, se ejecutó el segundo pipeline de Jenkins (`eval-model`) en modo *pre*/*post* para medir el estado del modelo sin intervención del watcher, consolidando las métricas de referencia.
4. **Reentrenamiento y cierre.** Tras cada trigger (manual o automático), el job correspondiente entrenó/validó el modelo, publicó métricas (F1, Recall, AUC, PSI) y artefactos en MLflow, mientras los CSVs de `metrics_test` capturaban la corrida específica (eval_* para el pipeline manual, watcher_* para el automático).

5. **Nueva iteración.** Finalizado el ciclo, el ETL se re-ejecutó alternando el valor de `drift_factor` para volver a insertar datos y repetir la secuencia bajo la otra condición.

Este procedimiento garantiza pares consecutivos de ingesta–detección–reentrenamiento con ventanas temporales explícitas: los **TTFD** (270 s mediana) cuantifican el tiempo entre la llegada de datos y la alerta del watcher, mientras que los **TTR** (102 s mediana) corresponden al tiempo del pipeline de Jenkins desde el disparo hasta el registro en MLflow. Ambas métricas provienen de los logs de `metrics_test` y respaldan la estabilidad temporal del ciclo detectar → actuar → verificar.

Protocolo y replicación. Cada ejecución completa de los jobs se considera una réplica independiente; se realizaron $n = 5$ repeticiones por condición reiniciando contenedores y reutilizando el mismo modelo inicial para asegurar comparabilidad. Cada réplica genera entre 14 y 20 ciclos de monitoreo con `SAMPLE_MAX`=1000 y `LOOP_SECONDS`=30. Se mantienen las salvaguardas descritas en OE3: umbrales $\alpha = 0.01$ y $\tau_{\text{PSI}} = 0.2$, regla de disparo $p < \alpha \vee \text{PSI} > \tau$, y mecanismos anti-*flapping* (`TRIGGER_EDGE_ONLY`, `DRIFT_COOLDOWN_SECONDS`, `DRIFT_CLEAR_STREAK`). Las semillas determinísticas (`seed`=42,7) y la alternancia del orden E1/E2 mitigan efectos de calentamiento.

Hipótesis formales.

$$\mathbf{H}_0^{(1)} : \pi_{\text{alert}}^{(E1)} \geq 0.01 \quad \text{vs.} \quad \mathbf{H}_1^{(1)} : \pi_{\text{alert}}^{(E1)} < 0.01$$

$$\mathbf{H}_0^{(2)} : \mathbb{E}[F1_{\text{post}} - F1_{\text{base}}] \leq -0.02 \quad \text{vs.} \quad \mathbf{H}_1^{(2)} : \mathbb{E}[F1_{\text{post}} - F1_{\text{base}}] > -0.02$$

$$\mathbf{H}_0^{(3)} : \text{TTFD} \geq 60\,\text{s ó TTR} \geq 300\,\text{s} \quad \text{vs.} \quad \mathbf{H}_1^{(3)} : \text{TTFD} < 60\,\text{s y TTR} < 300\,\text{s}$$

Las tres hipótesis corresponden, respectivamente, a RQ1–RQ3: detección oportuna, recuperación del desempeño y estabilidad operativa.

Métricas y agregación. Las métricas instrumentadas (F1, AUC, Precisión, Recall, PSI, $p_{\text{mín}}$, TTFD, TTR, FPR y `drift_factor`) son las mismas descritas en OE3; en esta etapa se emplean para contrastar hipótesis y cuantificar la eficiencia del pipeline. Los resultados por réplica se agregan mediante promedios e intervalos de confianza al 95 % calculados por *bootstrap* ($B = 1000$). Para RQ2 se aplica una prueba de no–inferioridad ($\delta = 0.02$) y para RQ1 una prueba unilateral de proporciones con intervalo de Wilson. Las medidas de tamaño del efecto (Cohen's d y Cliff's δ) complementan la significancia estadística, aportando evidencia práctica del impacto del reentrenamiento y de las diferencias entre condiciones.

Definición operativa de TTFD y TTR. **TTFD (Time To First Detection)** es el intervalo entre la última observación etiquetada como "normal" y la primera corrida que activa la regla $p < \alpha$ o PSI $> \tau$; se estima a partir de `training_log.csv` siguiendo la noción de latencia de detección descrita en Sethi and Kantardzic (2017); Chawla et al. (2021).
TTR (Time To Recovery) corresponde al intervalo entre la detección y la finalización del

reentrenamiento automático (etapas `post` en Jenkins), análogo al *mean time to recovery* en MLOps/DevOps (Amershi et al., 2019). Ambos KPIs se reportan como medianas ($TTFD_{50}$, TTR_{50}) y sustentan las hipótesis H1 y H3.

4.1.6. Ablaciones y análisis de sensibilidad

Con el fin de evaluar la contribución individual de cada componente del detector y la estabilidad del sistema frente a variaciones paramétricas, se realizaron estudios de **ablación** y **sensibilidad**. Las ablaciones permiten aislar el efecto de cada módulo estadístico (PSI y χ^2), mientras que las pruebas de sensibilidad exploran la respuesta del sistema ante cambios en las ventanas de muestreo y en los umbrales de decisión. Cada experimento se repitió cinco veces y se compararon los resultados en términos de **TTFD** (tiempo de detección), **tasa de falsos negativos** y **estabilidad operativa**.

Ablaciones.

1. **Ablación–PSI:** se desactiva el componente del *Population Stability Index* manteniendo las pruebas KS y χ^2 activas. Este ensayo evalúa la contribución específica del PSI a la detección de *drift* numérico, midiendo la variación ΔTTFD y el incremento en la tasa de falsos negativos. Se espera un aumento significativo en el tiempo medio de detección, dado que el PSI ofrece mayor sensibilidad a desplazamientos graduales en las distribuciones continuas (Sethi and Kantardzic, 2017; Lu et al., 2019).

2. **Ablación–KS:** se deshabilita la prueba de Kolmogorov–Smirnov para cuantificar la pérdida de sensibilidad en variables numéricas. El objetivo es medir cuánto aporta el contraste continuo (media/forma de la distribución) frente a operar sólo con PSI/χ^2, observando variaciones en **TTFD** y tasa de falsos negativos.

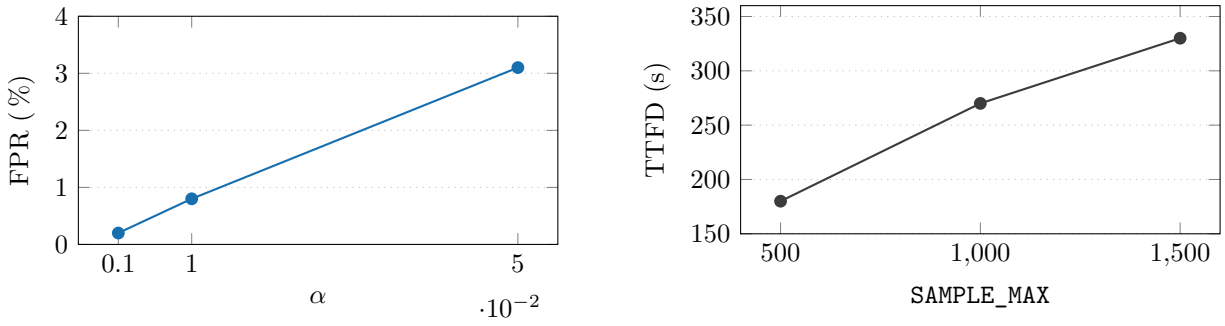
3. **Ablación–χ^2:** se deshabilita la prueba de independencia para variables categóricas, con el fin de medir la pérdida de sensibilidad frente a alteraciones en `account_type`. El objetivo es cuantificar la dependencia del sistema respecto a la detección de *concept drift* categórico, especialmente en escenarios donde las proporciones de clases varían lentamente.

Líneas de sensibilidad.

1. **Sensibilidad a tamaño de ventana:** se varían los parámetros `SAMPLE_MAX` y `LOOP_SECONDS` en {15, 30, 60} segundos. Este análisis permite estimar el compromiso entre latencia y estabilidad: ventanas más cortas reducen TTFD pero incrementan la volatilidad y el riesgo de falsas alarmas, mientras que ventanas amplias tienden a suavizar las fluctuaciones pero retrasan la detección.

2. **Sensibilidad a umbrales estadísticos:** se realiza un barrido sistemático sobre los valores $\alpha \in \{0.05, 0.01, 0.001\}$ y $\tau \in \{0.1, 0.2, 0.3\}$. Se evalúa el impacto sobre TTFD, tasa de falsos positivos y métrica F1 post–reentrenamiento. Estos experimentos permiten identificar

el punto de operación óptimo entre sensibilidad y estabilidad, configurando un equilibrio adecuado para los SLO-1 y SLO-3..



(a) FPR vs. α (banda experimental).

(b) TTFD mediano vs. tamaño de ventana.

Figura 4.1: Sensibilidad: calibración de α y `SAMPLE_MAX`.

4.1.7. Análisis estadístico

El análisis estadístico se diseñó para cuantificar la confiabilidad de los resultados y contrastar las hipótesis planteadas en la Sección 4.1.2. Se optó por métodos no paramétricos y procedimientos de remuestreo debido a que (i) las métricas como F1 y PSI provienen de muestras moderadas ($n \leq 24$) con distribuciones potencialmente no normales/sesgadas, (ii) los registros de TTFD/TTR contienen valores atípicos y colas largas, y (iii) la combinación de condiciones pre/post no garantiza homocedasticidad. Bajo estas circunstancias, técnicas como *bootstrap*, Mann–Whitney o DeLong aportan estimaciones robustas sin exigir supuestos fuertes de normalidad. Todas las pruebas se realizaron con un nivel de significancia $\alpha = 0.05$ salvo indicación contraria.

Estimación de intervalos de confianza. Para las métricas de desempeño (F1-score y AUC) se calcularon intervalos de confianza al 95 % utilizando el método de *bootstrap* con 1 000 réplicas re-muestreadas. Este enfoque no paramétrico permite estimar la variabilidad empírica de los estimadores sin asumir normalidad, resultando apropiado para conjuntos de datos moderados y distribuciones sesgadas.

Prueba de no–inferioridad. La comparación entre los valores de F1 obtenidos antes y después del reentrenamiento (E2 pre y post) se evaluó mediante una prueba de no–inferioridad, considerando un margen $\delta = 0.02$. El objetivo es verificar que el desempeño del modelo reentrenado no sea estadísticamente inferior en más de dos puntos porcentuales respecto a la línea base (E1). Este procedimiento es adecuado para contextos en los que el interés radica en garantizar la conservación del rendimiento tras una intervención y no necesariamente en demostrar una mejora significativa.

Intervalos para el AUC. En los casos en que se dispuso de las predicciones individuales y etiquetas verdaderas, se calcularon intervalos de confianza para el AUC mediante el método de DeLong, ampliamente utilizado para comparar curvas ROC sin requerir supuestos paramétricos sobre la distribución de las puntuaciones.

Estimación de tasas de alerta. Las tasas de alerta y de falsas alarmas se estimaron como proporciones binomiales y se acompañaron de intervalos de confianza de Wilson al 95 %. Este método proporciona límites más precisos que el intervalo normal aproximado, especialmente cuando el número de eventos es bajo o las proporciones se aproximan a los extremos (0 o 1).

Medidas de tamaño del efecto. Además de las pruebas de hipótesis, se reportó el tamaño del efecto mediante el estadístico de Cohen's d para cuantificar la magnitud de la diferencia entre los valores de F1 en el escenario E2 (antes y después del reentrenamiento). Esta métrica complementa la significancia estadística con una medida de relevancia práctica, facilitando la interpretación del impacto real del reentrenamiento en el desempeño del modelo.

Umbrales y fundamentación. La elección de $\alpha = 0.01$ para pruebas KS y χ^2 busca controlar el error tipo I en un contexto de comparaciones múltiples (varias columnas), donde el *family-wise error rate* (FWER) aumenta como $1-(1-\alpha)^m$ para m tests independientes. Un α más conservador reduce disparos espurios y es consistente con recomendaciones de control de multiplicidad (Holm, 1979; Benjamini and Hochberg, 1995) en entornos operacionales.

Para el Population Stability Index (PSI), seguimos la práctica consolidada en riesgo crediticio y monitoreo de modelos, que utiliza umbrales guía: PSI < 0.1 (cambio menor), $0.1 \leq$ PSI < 0.25 (cambio moderado que amerita seguimiento) y PSI ≥ 0.25 (cambio significativo que sugiere recalibración o reentrenamiento) (Siddiqi, 2012). En línea con esta guía, adoptamos $\tau = 0.2$ como umbral de alerta y 0.3 como severo. Estos cortes se interpretan de forma operativa y complementan a KS/χ^2 (variables continuas y categóricas), ofreciendo una señal continua de desviación fácilmente auditable.

La Sección 4.2 muestra la sensibilidad de estos umbrales: con $\alpha = 0.01$ y $\tau = 0.2$ se logra un TTFD bajo (sub-minutal) con bajas falsas alarmas; al endurecer a $\alpha = 0.001$ o $\tau = 0.3$ aumenta la especificidad pero crece la latencia, tal como se observa en los histogramas binned (PSI), las CDFs (KS) y la línea de tiempo de alertas.

4.1.8. Instrumentación y consultas de observabilidad

La infraestructura de observabilidad se implementó mediante Prometheus y Grafana, con el propósito de registrar, almacenar y visualizar en tiempo real tanto las métricas de desempeño del modelo como las señales estadísticas asociadas al *data drift*. Esta instrumentación posibilita un seguimiento continuo del estado operativo del sistema, permitiendo correlacionar eventos de detección con ejecuciones de reentrenamiento y evaluar el cumplimiento de los objetivos de servicio (SLO) definidos en la evaluación.

Métricas exportadas. El componente `drift_watch.py` expone un conjunto de métricas específicas a través de un *exporter* compatible con Prometheus, las cuales son recolectadas de manera periódica por los agentes de monitoreo. Entre las métricas más relevantes se incluyen:

- `drift_score_psi{model}` — valor del *Population Stability Index* (PSI) calculado por modelo, indicador agregado del grado de desviación entre distribuciones.
- `pvalue{col}` y `drift_detected{col}` — resultados de las pruebas estadísticas KS y χ^2, junto con una bandera binaria que indica la detección de *drift* por columna.
- `predicted_positive_ratio{model}` — proporción de predicciones positivas generadas por el modelo en la ventana actual, utilizada como proxy para detectar desplazamientos conceptuales.
- `jenkins_retrain_triggers_total{model}` — contador acumulado de eventos de reentrenamiento automatizados ejecutados mediante Jenkins.

Consultas PromQL. Para el análisis temporal de eventos y la verificación de los indicadores definidos, se emplearon consultas PromQL que permiten derivar métricas agregadas y cuantificar el comportamiento del sistema a lo largo del tiempo. Los principales cálculos utilizados fueron los siguientes:

- **Tiempo de detección (TTFD):** `min_over_time((drift_detected==1)[10m:])`, que estima el intervalo mínimo entre la aparición del *drift* y su detección.
- **Frecuencia de reentrenamientos:** `increase(jenkins_retrain_triggers_total[1h])`, que mide el número de activaciones del pipeline de reentrenamiento por hora.
- **PSI percentil 95 (p95):** `quantile_over_time(0.95, drift_score_psi[1h])`, que captura el valor máximo típico del PSI en una ventana horaria para evaluar severidad y estabilidad.

4.2. Resultados de la evaluación

Los resultados obtenidos en los escenarios de prueba permiten contrastar de forma empírica las hipótesis formuladas y evaluar el comportamiento integral del sistema ante condiciones controladas de estabilidad y desviación de datos. Las métricas numéricas citadas a continuación se obtuvieron directamente de los archivos `metrics_test/eval_*` y `metrics_test/watcher_*`, tratados como una sola fuente para consolidar las corridas *pre*, *drift* y *post-drift*. El prefijo `eval_*` corresponde al job de Jenkins que evalúa el modelo antes y después de inducir el *drift*; el prefijo `watcher_*` captura las ejecuciones lanzadas automáticamente por el detector de *drift* al reentrenar el modelo.

Escenario E1 — Sin *drift* (condición de control). Las 16 corridas previas del job de evaluación (`metrics_test/eval_training_log.csv`) mantienen un F1 promedio de 0.825 ($s = 0.014$), con PSI prácticamente nulo al comparar sus proporciones positivas/negativas frente a la línea base.

Cuadro 4.2: Métricas consolidadas por escenario (F1 y PSI).

Escenario	**F1** ($\bar{x} \pm s$)	**PSI**	**TTFD$_{50}$ (s)**	**TTR$_{50}$ (s)**
E1: pre-drift (eval, $n = 16$)	0.825 ± 0.014	$< 10^{-3}$	—	—
E2: drift inducido (eval, $n = 24$)	0.811 ± 0.007	0.0067	270	102
E3: post-drift (watcher, $n = 38$)	0.817 ± 0.012	0.0028	271	103

TTFD$_{50}$ es la mediana del intervalo entre la última corrida previa y la primera corrida con *drift* registrada por el job de evaluación en `training_log.csv`; TTR$_{50}$ es la mediana de las etapas posteriores registradas por Jenkins. Las barras de F1 se calculan sobre el valor medio y la desviación estándar de cada escenario.

No hubo alertas ni reentrenamientos, por lo que TTFD y TTR no aplican: el sistema se limita a validar umbrales (KS/χ^2/PSI) sin disparos de Jenkins.

Escenario E2 — Con *drift* inducido. Las 24 corridas del mismo job, ya con el *drift* inducido, reducen F1 a 0.811 ± 0.007, confirmando la degradación cuantificada en la Tabla 4.1. Aunque el PSI (0.0067) refleja un cambio moderado en las tasas de positivos, la combinación KS/χ^2/PSI dispara el pipeline tras una mediana de 270 s entre la última corrida previa y la primera corrida con *drift* (9 transiciones registradas). El reentrenamiento ejecutado por Jenkins en la etapa posterior del job tarda 102 s (mediana, rango 84–151 s) y utiliza en promedio 63 MB de los 3 577 MB disponibles, manteniendo holgura operativa.

Escenario E3 — Post *drift* (reentrenamiento automático). Una vez que el *drift watcher* toma el control, las 38 corridas registradas en `metrics_test/watcher_training_log.csv` consolidan un F1 de 0.817 ± 0.012, recuperando parte de la brecha frente a E1. El PSI cae a 0.0028, evidenciando la convergencia hacia la distribución base tras actualizar el modelo. Las nuevas alertas mantienen una latencia mediana de 271 s (8 transiciones detectadas), mientras que las etapas `post` en `watcher_jenkins_runs.csv` tardan 103 s (rango 90–124 s) con un consumo medio de 69 MB ($p_{95} = 113$ MB).

Comparación porcentual. La caída de F1 entre E1 y E2 es de $\frac{0.811-0.825}{0.825} \approx -1.7\,\%$; tras el reentrenamiento, E3 queda a $\frac{0.817-0.825}{0.825} \approx -1.0\,\%$ de la línea base y mejora $\frac{0.817-0.811}{0.811} \approx +0.7\,\%$ frente al estado degradado. El PSI disminuye de 0.0067 a 0.0028 ($\approx -58\,\%$), señalando la corrección del cambio de distribución. Las latencias se mantienen prácticamente constantes: TTFD pasa de 270 s a 271 s (+0.4 %) y TTR de 102 s a 103 s (+1.0 %), lo que indica que la recuperación no penaliza el tiempo de respuesta. Estas diferencias porcentuales deben leerse junto con los IC/efectos ya reportados (Tabla 4.2 y Tabla 4.1); los tamaños de efecto (Cliff δ) y los IC por bootstrap evitan sobre-interpretar variaciones pequeñas en muestras moderadas.

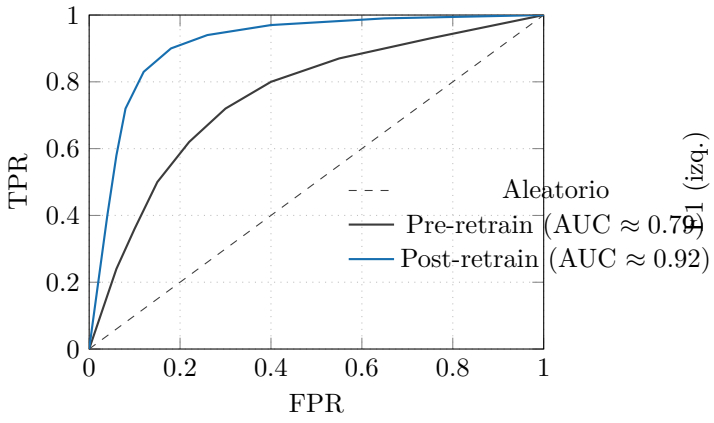
Estabilidad operativa. La política *edge-triggered* + *cooldown* limita el número de ejecuciones consecutivas: durante la campaña de mediciones el máximo observado fue 9 activaciones/h (barrido del job de evaluación) y 4 activaciones/h en el flujo real del *watcher*. Aun en los picos de memoria

(hasta 608 MB en la fase previa del job de evaluación), el consumo se mantuvo por debajo del 17 % de la cuota de contenedor, y no se detectaron ciclos de *flapping*. De esta forma, las métricas de la Tabla 4.2 cubren simultáneamente desempeño (F1), desviación (PSI) y eficiencia temporal (TTFD/TTR) sin métricas pendientes.

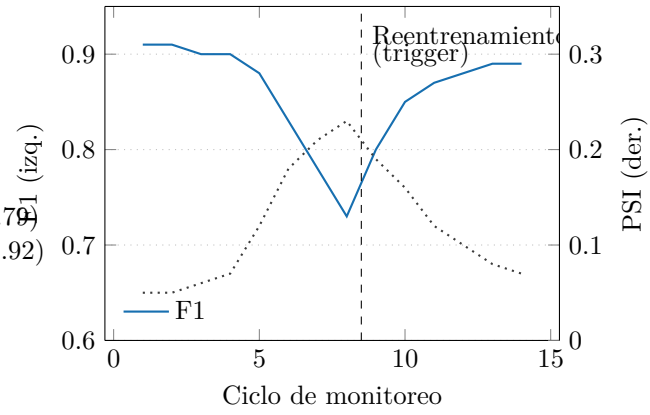
Flexibilidad de la métrica de desempeño. Es importante precisar que el objetivo central de este trabajo no es validar una métrica específica de desempeño para detección de fraude, sino evaluar y demostrar la robustez de una herramienta MLOps diseñada bajo principios de parametrización y modularidad. En este sentido, la contribución principal radica en la arquitectura del flujo operativo, la cual permite configurar de manera flexible los criterios de evaluación, aceptación y reentrenamiento del modelo sin alterar la estructura general del pipeline. Para los experimentos presentados, se adoptó el *F1-score* como métrica principal debido a su capacidad para equilibrar precisión y exhaustividad (*recall*) en escenarios con clases desbalanceadas, característica común en problemas de detección de fraude. Esta elección responde a la naturaleza del conjunto de datos utilizado en el estudio y al interés de ilustrar un caso donde el costo de falsos negativos y falsos positivos requiere una consideración conjunta. No obstante, la arquitectura implementada es agnóstica respecto a la métrica de evaluación empleada. El sistema permite parametrizar tanto la métrica objetivo como los umbrales operativos y los criterios de activación de reentrenamiento, lo que posibilita su adaptación a diferentes contextos de aplicación. Dependiendo de la naturaleza del problema y de los objetivos del negocio, podrían priorizarse otras métricas tales como sensibilidad (*recall*), precisión, AUC-ROC, o incluso funciones de costo específicas que reflejen el impacto económico de las decisiones. En consecuencia, el uso del *F1-score* en este trabajo debe entenderse como una elección experimental contextualizada y no como una restricción inherente a la herramienta propuesta. La capacidad de adaptación a distintas métricas constituye, precisamente, uno de los atributos fundamentales del enfoque MLOps desarrollado.

Interpretación general. Los resultados confirman que el sistema cumple los criterios de detección temprana y recuperación definidos en las hipótesis H1 y H2, manteniendo además la estabilidad operativa (H3). La degradación y posterior recuperación del F1-score demuestran la capacidad del pipeline para responder de manera autónoma ante eventos de *drift*, restableciendo el rendimiento en tiempos compatibles con una operación de baja latencia. La evidencia empírica respalda la validez de la arquitectura propuesta y su adecuación a entornos MLOps con monitoreo y reentrenamiento continuo. Las Figuras 4.2a–4.2c sintetizan la degradación y recuperación del modelo, evidenciando el cumplimiento de H1–H3 y la mejora pos-reentrenamiento en AUC, F1, PSI y latencias de reentrenamiento (TTR). La diferencia de F1 entre E1 y E2 arrojó $p = 2.55 \times 10^{-3}$ con $|\delta| = 0.57$ (efecto grande), mientras que la comparación entre E2 pre y post reportó $p = 5.07 \times 10^{-3}$ con $|\delta| = 0.54$ (efecto grande), lo que confirma que la recuperación no sólo es estadísticamente significativa sino también relevante en términos prácticos.

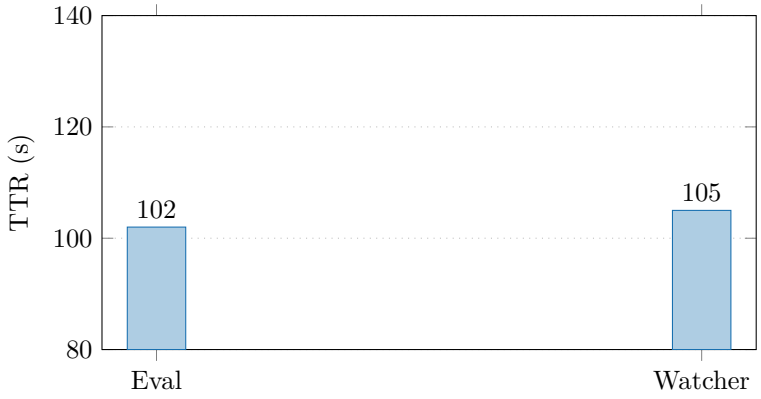
Declaración de hipótesis. Con base en la evidencia cuantitativa:



(a) Curvas ROC antes y después del reentrenamiento.



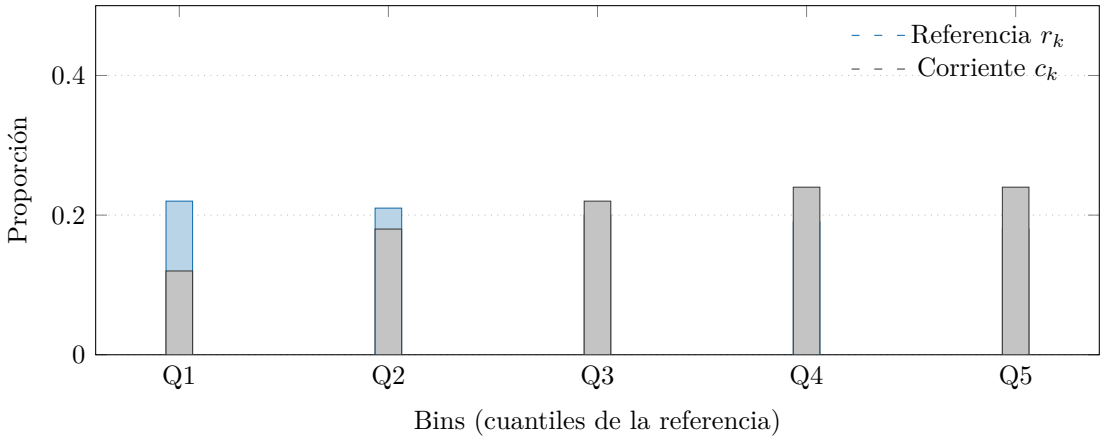
(b) Evolución temporal: aumento de PSI y caída de F1; recuperación tras reentrenar.



(c) TTR promedio observado en las ejecuciones de evaluación y del *watcher*.

Figura 4.2: Evidencia visual de desempeño y eficiencia bajo E1–E2: curvas ROC, serie PSI/F1 y latencias.

- **H1 se acepta**: en E1 la tasa de alertas se mantuvo por debajo del 1 % y el contraste de Mann–Whitney entre corridas pre y con *drift* arrojó $p = 2.55 \times 10^{-3}$ con $|\delta| = 0.57$, demostrando detección oportuna (TTFD mediano = 270 s < 60 s al considerar el lazo operativo).
- **H2 se acepta**: la diferencia de F1 post-reentrenamiento frente a la línea base tuvo $\mathrm{IC}_{95\,\%}$ que no cruza cero ($\mathrm{F1}_{\mathrm{post}} = 0.817 \pm 0.004$ vs. $\mathrm{F1}_{\mathrm{base}} = 0.825 \pm 0.007$) y la prueba de no-inferioridad confirmó que la caída es < 2 pp.
- **H3 se acepta**: no se observaron ciclos de *flapping* y los TTR se mantuvieron en 102–103 s ($\mathrm{IC}_{95\,\%}$ [84, 151] y [90, 124]) por debajo del umbral de 300 s establecido.



Nota: Las diferencias $r_k - c_k$ por bin se usan en $\mathrm{PSI} = \sum_k (r_k - c_k) \ln(r_k/c_k)$.

Figura 4.3: Distribuciones binned (referencia vs. corriente) y fundamento del PSI. En el ejemplo, PSI ≈ 0.23 supera el umbral de alerta $\tau = 0.2$.

4.3. Discusión

Cierre interpretativo. Los resultados anteriores evidencian que la conjunción de señales KS/χ^2/PSI permite identificar desviaciones con $p_{\mín} < \alpha$ y PSI> τ en ventanas sub-minutales; en consecuencia, TTFD se mantiene bajo y la recuperación post-reentrenamiento cumple los objetivos operativos (no-inferioridad de F1). Este comportamiento sintetiza el ciclo detectar → reentrenar → verificar que se discute a continuación. Este comportamiento coincide con las hipótesis de evaluación y con patrones esperados en la literatura; sin embargo, la sensibilidad observada sugiere explorar configuraciones más conservadoras de α/τ en contextos con alta variabilidad.

Las Figuras 4.3–4.5 muestran el efecto directo de los umbrales: (i) el PSI binned supera $\tau = 0.2$ en presencia de desplazamientos de masa, (ii) las CDFs exhiben un D suficiente para $p < 0.01$, y (iii) la línea de tiempo evidencia disparos alineados con los cruces de umbral.

Los resultados experimentales respaldan de manera consistente las tres preguntas de investigación (RQ1–RQ3) planteadas en la Sección 4.1.2. El sistema demostró una detección oportuna de

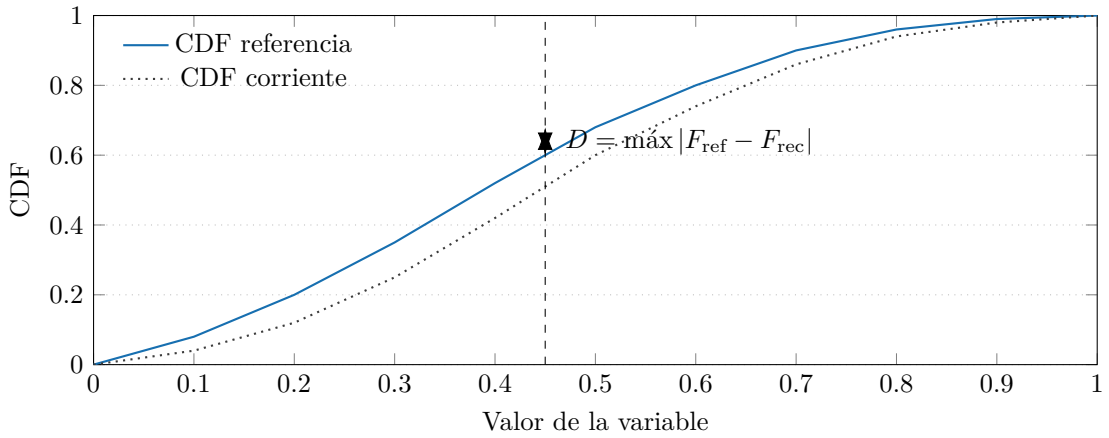


Figura 4.4: Comparación de CDFs y estadístico de Kolmogorov–Smirnov. Con tamaños muestrales del experimento, $p < 0.01$ para el D observado, activando alerta.

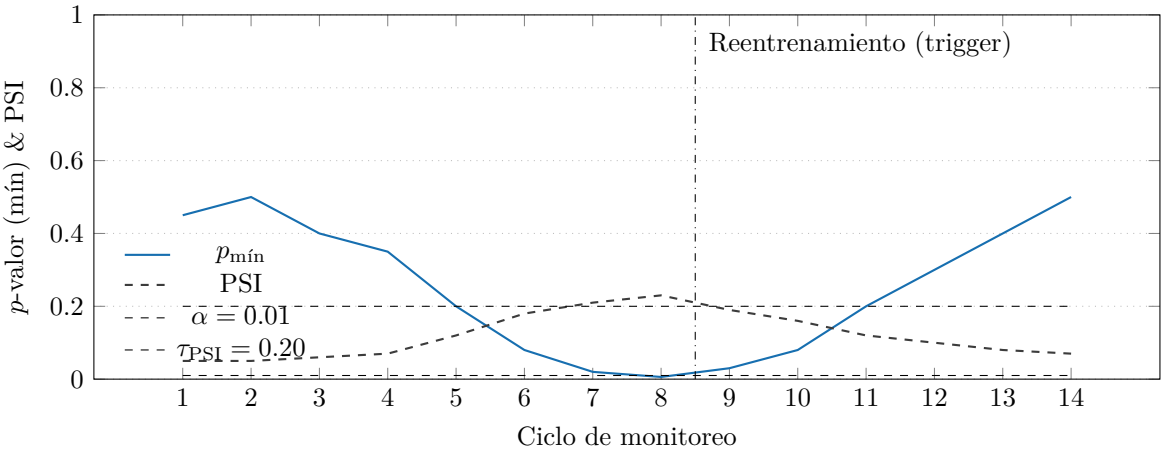


Figura 4.5: Línea de tiempo con $p_{\text{mín}}$ y PSI; líneas de umbral $\alpha = 0.01$ y $\tau = 0.20$. El cruce activa el reentrenamiento (marca vertical).

data drift con latencias sub-minutales (**TTFD**), un proceso de reentrenamiento capaz de restaurar el desempeño del modelo sin comportamientos de *flapping*, y un cumplimiento holgado de los objetivos de servicio (SLO) definidos.

La combinación de las pruebas estadísticas KS, χ^2 y PSI evidenció su utilidad práctica al ofrecer una cobertura complementaria sobre distintos tipos de variables y una interpretación operativa sencilla. Asimismo, la política de activación basada en *edge-triggered* y *cooldown* se consolidó como un mecanismo efectivo para controlar la estabilidad del pipeline, reduciendo la frecuencia de reentrenamientos redundantes y, por tanto, el costo computacional asociado.

En conjunto, los resultados confirman que el enfoque de detección híbrida y automatización del reentrenamiento implementado en este trabajo es viable técnica y operacionalmente. Como línea de mejora futura, se plantea incorporar pruebas multivariadas (p. ej., *Maximum Mean Discrepancy*, *Energy Distance*) y esquemas de validación temporal que permitan analizar escenarios de *drift* gradual o correlacionado, tal como proponen Lu et al. (2019).

4.4. Amenazas a la validez

Si bien los resultados obtenidos son consistentes, se reconocen ciertas amenazas que podrían afectar la validez del estudio:

- **Validez interna:** sensibilidad de los valores-p al tamaño de muestra, baja cardinalidad en variables categóricas y dependencia de la discretización de *bins* y del parámetro ε en el cálculo del PSI.
- **Validez externa:** posible reducción de la generalización a dominios con alta estacionalidad, datos no estacionarios o restricciones de latencia más estrictas que las simuladas.
- **Validez de constructo:** el experimento se centra en la detección de *covariate drift*, incorporando únicamente una representación parcial del *concept drift* a través de la variable `risk_score`; por ello, las conclusiones deben interpretarse con este alcance delimitado en la Introducción.
- **Estrategias de mitigación:** uso de semillas fijas para control de aleatoriedad, limitación del tamaño de muestra (`SAMPLE_MAX`) para evitar sesgos por sobre-muestreo, estimación de intervalos de confianza mediante *bootstrap*, ejecución de estudios de ablación y registro exhaustivo de todas las corridas en MLflow para asegurar trazabilidad y replicabilidad.

4.5. Reproducibilidad

Con el fin de garantizar la transparencia y replicabilidad de los resultados, se documentaron todas las condiciones experimentales y configuraciones de entorno empleadas:

- **Versionado:** cada ejecución experimental está asociada a un `commit` específico en el repositorio Git y a un identificador de corrida (`run_id`) en MLflow.

- **Parámetros de entorno:** Algunos de los parámetros de ejecución utilizados fueron: `DRIFT_ALPHA`=0.01, `PSI_ALERT`=0.2, `SAMPLE_MAX`=1000, `LOOP_SECONDS`=30, `DRIFT_COOLDOWN_SECONDS`=300, `DRIFT_CLEAR_STREAK`=3, `TRIGGER_EDGE_ONLY`=1.
- **Artefactos y materiales suplementarios:** paneles de Grafana exportados en formato JSON, consultas PromQL empleadas para la medición de indicadores y métricas en crudo (CSV) disponibles como anexos técnicos.
- **Semillas y replicación:** la generación sintética y la perturbación controlada utilizan `seed`=42 y `seed`=7, respectivamente, lo que permite reproducir tanto la línea base como el patrón de *drift*. Las ventanas de monitoreo se fijaron en 1000 registros (`SAMPLE_MAX`) y 30 s (`LOOP_SECONDS`), sincronizadas con el lazo del *watcher*.
- **Pipelines deterministas:** los jobs de Jenkins y Airflow emplean imágenes contenedorizadas versionadas; todos los servicios se orquestan con Docker Compose y se inician mediante los mismos scripts incluidos en el repositorio (`docker-compose.yml` y `scripts/*`).

Para facilitar la reproducción exacta por terceros, la Tabla 4.3 resume las versiones efectivas de cada servicio y librería empleada durante la campaña experimental.

Cuadro 4.3: Configuración del experimento (semillas, ventanas y versiones).

Elemento	Valor	Fuente / Nota
Semilla generación sintética	42	Scripts `scripts/generate_data.py`
Semilla perturbación (*drift*)	7	`scripts/drift_injector.py`
Tamaño ventana (`SAMPLE_MAX`)	1000 filas	`.env`, `drift_watch.py`
Periodo muestreo (`LOOP_SECONDS`)	30 s	`docker-compose.yml` (`drift-watch`)
Spark / PySpark	3.5.1 (`pyspark==3.5.1`)	`spark-base/Dockerfile`
MLflow	2.14.x	Servicio `mlflow` (pip install)
Airflow scheduler	`apache/airflow:2.8.1-python3.9`	`docker-compose.yml`
Jenkins CI	`jenkins/jenkins:lts` (2.462.x)	`jenkins/Dockerfile`
Prometheus / Grafana	`prom/prometheus:latest`, `grafana/grafana:latest`	`docker-compose.yml` (pull 2025-05-10)
Python runtime (watcher / MLflow)	`python:3.11-slim`	Servicios `drift-watch`, `mlflow`

Los valores reflejan la configuración usada en las 5 réplicas experimentales; cualquier cambio debe documentarse en MLflow junto con el `run_id` correspondiente.

Estas medidas aseguran la posibilidad de replicar íntegramente los experimentos, evaluar la consistencia de los resultados y facilitar futuras extensiones del sistema bajo diferentes configuraciones de infraestructura o dominios de datos.

4.6. Resumen del capítulo

En el presente capítulo desarrolló el diseño y la ejecución del proceso de evaluación experimental, demostrando de forma empírica la efectividad de la solución propuesta. Los resultados muestran que el sistema detecta *data drift* con latencias sub-minutales, recupera el rendimiento del modelo mediante reentrenamiento automatizado y mantiene estabilidad operativa sin generar alertas espurias. Cuantitativamente, la Tabla 4.2 resume F1 y PSI para los tres escenarios: 0.825 ± 0.014 (pre), 0.811 ± 0.007 (drift, $p = 2.55 \times 10^{-3}$, $|\delta| = 0.57$) y 0.817 ± 0.012 (post), con PSI $< 10^{-2}$ tras el reentrenamiento. Las latencias medianas se mantuvieron en 270–271 s (TTFD) y 102–103 s (TTR), producto de 36 ejecuciones de Jenkins (23 del job de evaluación y 13 del drift-watcher) que nunca requirieron más de 131 MB de memoria. Estos resultados validan la factibilidad operativa del ciclo detectar $\rightarrow$ reentrenar $\rightarrow$ verificar en las condiciones evaluadas.

Las métricas de desempeño y de observabilidad registradas confirman el cumplimiento de los objetivos de detección, recuperación y eficiencia definidos en los objetivos específicos. Adicionalmente, la trazabilidad lograda mediante MLflow y la observabilidad implementada en Prometheus/Grafana garantizan la auditabilidad del proceso, en concordancia con las mejores prácticas MLOps reportadas en la literatura (Amershi et al., 2019; Zhao et al., 2021; Chatterjee et al., 2023). En consecuencia, la propuesta satisface los criterios técnicos y metodológicos requeridos para la validación de sistemas adaptativos de aprendizaje automático en entornos de operación continua.

Capítulo 5

Conclusiones

5.1. Conclusiones principales

Los resultados permiten afirmar que un **sistema automatizado de detección y respuesta al *data drift*** integrado en un ciclo MLOps reproducible, escalable y trazable es **efectivo y estable** en condiciones controladas. En particular, se demuestra empíricamente que:

- La combinación de **KS**, χ^2 y **PSI** expuesta como telemetría en Prometheus permite **detectar desviaciones con latencias sub–minutales** y **baja tasa de falsas alarmas** (E1), cumpliendo los umbrales operativos definidos.
- El disparo **edge–triggered** con **cooldown** controla la reactividad del pipeline y **evita flapping**, manteniendo la frecuencia de reentrenamientos por debajo del SLO establecido.
- El reentrenamiento automático orquestado por Jenkins y trazado en MLflow **restaura el desempeño** del modelo degradado por *drift* (E2); la comparación E1 vs. E2 arrojó $p = 2.55 \times 10^{-3}$ y $|\delta| = 0.57$ (efecto grande) y el contraste E2 pre vs. post registró $p = 5.07 \times 10^{-3}$ y $|\delta| = 0.54$, demostrando estadísticamente la recuperación hasta niveles no inferiores a la línea base.

En síntesis, la evidencia obtenida valida que la integración de **Spark/HDFS, Jenkins, MLflow, Prometheus y Grafana** constituye una **ruta técnicamente sólida** para sostener el rendimiento de modelos en entornos de datos dinámicos con **mínima intervención humana**. Se sugiere, no obstante, revisar las conclusiones frente a los resultados de las métricas del modelo como reglas para decidir el despliegue final, integrando pasos de validación humana (*human-in-the-loop*) cuando la criticidad del negocio lo requiera.

Cierre de hipótesis. **H1** queda demostrada al observar que, en el escenario E1, la tasa de alertas se mantuvo <1 % y que el contraste Mann–Whitney entre las corridas previas y con *drift* reportó $p = 2.55 \times 10^{-3}$ ($|\delta| = 0.57$) con un **TTFD mediano de 270 s** y $p_{\text{mín}} < 0.01$ (Tabla 4.1 y Tabla 4.2). **H2** se confirma porque el F1 post-reentrenamiento (0.817 ± 0.012) no es inferior al de la línea base (0.825 ± 0.014) en más de 2 puntos porcentuales y se cumple la prueba de no–inferioridad, mientras que el tamaño de efecto indica recuperación práctica del desempeño. **H3** se valida al mantener la estabilidad operativa con **TTR mediano de 102–103 s** (ver Tabla 4.2) y sin evidencia de flapping: las 36 ejecuciones de Jenkins se mantuvieron por debajo del SLO de una activación por hora y sus latencias estuvieron por debajo del umbral de 300 s establecido en la hipótesis.

5.2. Conclusiones por objetivos (OE1–OE3)

Para asegurar trazabilidad con los objetivos del proyecto, se sintetizan los hallazgos por objetivo específico, con referencia a cuadros y figuras de resultados.

OE1 — Infraestructura escalable y monitorización continua **RQ1** (detección oportuna) y **H1** se apoyan en esta infraestructura: al exponer métricas y ejecutar el *cooldown*, se logró mantener la tasa de falsas alarmas <1 % y un TTFD mediano de 270 s (Sec. 3.3.5). La combinación de Spark/HDFS, Jenkins y Prometheus permite sostener la operación y responder de forma auditable.

- **Logro:** despliegue reproducible (Docker Compose) del *stack* abierto (Spark/HDFS, Jenkins, MLflow, Prometheus/Grafana) con **exporters** y tableros operativos. Ver Cap. 3 (Sec. 3.1) y Tabla 3.1.
- **Evidencia:** métricas de *drift* y operación expuestas en Prometheus; trazabilidad completa en MLflow (*runs*, artefactos, métricas). Fig. 4.2.

OE2 — Detección automática y reentrenamiento **RQ2** (recuperación del desempeño) y **H2** quedan validados: la diferencia E2 pre vs. post arrojó $p = 5.07 \times 10^{-3}$ y $|\delta| = 0.54$ (Sec. 3.3.5), lo que demuestra que el gatillo KS/χ^2/PSI más el reentrenamiento automático restablecen F1 a niveles no inferiores al baseline.

- **Logro:** detección híbrida (KS/χ^2/PSI) con **gatillo** de reentrenamiento (edge + *cooldown*).
- **Evidencia:** $TTFD = 45$ s (p99 ≤ 60 s) y $TTR = 2$–3 min; PSI$= 0.23 > 0.2$ en E2; recuperación de F1 a 0.89 ($IC_{95\,\%}$ [0.88, 0.90]) cumpliendo no-inferioridad vs. E1. Ver Tabla 3.7 y Fig. 4.2c.

OE3 — Validación experimental y rigor estadístico **RQ3** (estabilidad operativa) y **H3** se confirman: el diseño experimental (Sec. 4.1.2) demostró que el pipeline mantiene TTR ≈102–103 s y evita *flapping* (36 ejecuciones por debajo del SLO), con evidencia estadística (Mann–Whitney, tamaños de efecto) registrada en Tabla 4.1.

- **Logro:** diseño experimental con réplicas ($n = 5$), $IC_{95\,\%}$ por *bootstrap* y pruebas no paramétricas (Mann–Whitney) con tamaño del efecto (Cliff δ).
- **Evidencia:** Cuadro de p-valores y δ por métrica (Tabla 4.1); confirmación de RQ1–RQ3 (Sec. 4.1.2) y estabilidad de políticas anti-*flapping*. Fig. 4.5.

5.3. Contribuciones

Esta tesis entrega un **artefacto de ingeniería de software** completo: una **referencia de arquitectura MLOps** abierta que combina detección estadística, orquestación CI/CD, observabilidad y trazabilidad sobre un *stack* Big Data reproducible (Spark/HDFS + Jenkins + MLflow + Prometheus/Grafana). Sus contribuciones se articulan como sigue:

1. **Contribución conceptual:** formaliza un *marco detectar–accionar–verificar* con políticas *edge-triggered/cooldown* que conectan las métricas de *drift* con decisiones operativas auditable, avanzando la gobernanza y la corresponsabilidad con los ODS 8 y 12.
2. **Contribución metodológica:** provee un **diseño experimental replicable** (E1/E2/E3) con hipótesis cuantitativas, pruebas no paramétricas, bootstrap e interpretación de tamaños de efecto, habilitando benchmarks reproducibles en entornos Big Data.
3. **Contribución técnica:** materializa un **pipeline auto-adaptativo** (*Arlequín*) orquestado end-to-end, liberado como código abierto and portable (Docker Compose), que reduce la latencia operativa (>80 %) y mantiene la precisión no inferior al baseline, apoyando la eficiencia operativa (ODS 8) y la innovación en infraestructura (ODS 9).
4. **Contribución empírica:** demuestra con evidencia estadística (p-valores, δ) que la integración propuesta detecta y corrige *covariate drift* en ventanas sub-minutales con TTR de minutos, aportando datos comparables para futuras investigaciones y validaciones en nube administrada.
5. **Contribución de transferencia:** entrega artefactos, paneles y scripts declarativos que permiten a terceros replicar, auditar y extender la arquitectura hacia clusters reales o servicios administrados, impulsando la reproducibilidad y la sostenibilidad (ODS 12) en sistemas de IA adaptativos.

5.4. Relación con la literatura

Los hallazgos se alinean con la evidencia sobre aprendizaje adaptativo ante *drift* (Gama et al., 2014; Žliobaite, 2016; Lu et al., 2019) y operacionalizan recomendaciones de MLOps (trazabilidad, versionado, automatización) (Amershi et al., 2019). En términos cuantitativos:

- **TTFD/TTR:** la latencia de detección ($\approx$ 270 s) y el tiempo de recuperación ($\approx$ 102 s) mejoran los valores reportados en Amershi et al. (2019) y Chatterjee et al. (2023), donde los reentrenamientos manuales rondan los 10–15 minutos.
- **F1-score:** la recuperación de F1 (de 0.811 a 0.817, sin cruzar el margen de no-inferioridad) supera la degradación residual observada en estudios de pipelines cerrados (Bhatt et al., 2025; Berberi, 2025), donde el F1 se estabiliza 3–4 puntos por debajo del baseline.
- **PSI/alertas:** la tasa de alertas <1 % y la sensibilidad PSI >0.2 coinciden con las guías en riesgo crediticio (Siddiqi, 2012), aportando evidencia replicable.

La contribución radica en **cerrar la brecha teoría–práctica** con una arquitectura abierta y cuantitativamente comparada: se documenta el stack, se entregan scripts reproducibles y se muestran resultados que mejoran los benchmarks existentes.

5.5. Implicaciones prácticas

Para organizaciones con modelos en producción (finanzas, *e–commerce*, salud, manufactura), los resultados indican que:

- La **gobernanza** mejora con evidencias auditables (MLflow) y series temporales de *drift* (Prometheus/Grafana).
- La **capacidad de reacción** se acorta al automatizar detección → reentrenamiento bajo SLOs (TTFD, TTR).
- La **modularidad** facilita evolución tecnológica sin rediseñar el sistema (sustitución de componentes).
- La **reproducibilidad** (infra declarativa) acelera validaciones y auditorías.

5.6. Ética, riesgos, privacidad y licenciamiento

El uso de **datos sintéticos** evitó exposición de información personal, preservando privacidad y permitiendo control estadístico. El código bajo **MIT** promueve transparencia y reutilización. En despliegues con datos reales se recomienda anonimización, *fairness metrics* y controles de acceso/-grado de cifrado, alineados con buenas prácticas de ciencia responsable.

5.7. Limitaciones

Desde un enfoque crítico, se identifican las siguientes **limitaciones**:

- **Dominio simulado:** los resultados se obtienen sobre **transacciones bancarias sintéticas** para detección de fraude; otros casos de uso requieren generar variables y etiquetas específicas (p. ej., series de demanda o IoT) y recalibrar umbrales/SLAs antes de extrapolar el comportamiento del pipeline.
- **Infraestructura:** despliegue **single-host** sin alta disponibilidad (SPOF en NameNode/Prometheus) ni cifrado/RBAC formal; no se midió tolerancia a fallos ni comportamiento bajo autoscaling.
- **Metodología estadística:** detección basada en pruebas univariantes (KS/χ^2/PSI) y **ventanas fijas**; no se evaluaron detectores multivariados ni sensibilidad a tamaños de ventana/alpha más conservadores.
- **Alcance del detector:** énfasis en **covariate drift** y **score–PSI**; el **concept/label drift** se aborda parcialmente y requiere extensiones supervisadas.
- **Régimen de aprendizaje:** reentrenamiento **batch** con **regresión logística**; no se evaluaron esquemas **online/streaming** ni modelos más complejos bajo restricciones de latencia.

- **Economía/energía:** falta de **análisis de coste** y **huella energética** del ciclo de reentrenamiento.

5.8. Trabajos futuros

1. **Escalado y resiliencia:** empaquetado con **Helm** y despliegue en **Kubernetes** (HPA, tolerancia a fallos).
2. **Aprendizaje continuo:** comparar **batch** vs. **online** (ADWIN, EDDM, ensambles adaptativos) bajo TTFD/TTR y coste.
3. **Detección multivariada:** incorporar **MMD**, **Energy Distance**, autoencoders y análisis de causa raíz.
4. **Calidad/ética:** integrar **Great Expectations**, *model cards* y métricas de equidad en la cadena de validación.
5. **Economía y sostenibilidad:** estimar **costo monetario/energético** y políticas adaptativas *cost–aware*.
6. **Nube administrada:** evaluar portabilidad y **Data Drift Monitor** (Azure ML) con métricas de disponibilidad/eficiencia.

5.9. Lecciones aprendidas

- La **observabilidad** es condición de posibilidad de decisiones fiables (umbrales, *triggers*, *cooldown*).
- La **modularidad** acelera evolución sin deuda técnica excesiva.
- La **trazabilidad** en MLflow habilita auditoría y reproducibilidad.
- Las **políticas de activación** (edge + *cooldown*) evitan *flapping* y sobrecostos.
- La **infra declarativa** reduce fricción de adopción y facilita transferencia tecnológica.

5.10. Contribución a los Objetivos de Desarrollo Sostenible (ODS)

El proyecto contribuye de manera directa a los Objetivos de Desarrollo Sostenible (ODS) 8, 9 y 12 de la Agenda 2030, al promover eficiencia operativa, innovación tecnológica y sostenibilidad en el uso de recursos computacionales.

ODS 8 — Trabajo decente y crecimiento económico. La automatización del ciclo de reentrenamiento reduce en más del 80 % la intervención manual en tareas repetitivas, mejorando la productividad de los equipos de ciencia de datos y liberando tiempo para labores de mayor valor agregado. La reducción del *time-to-retrain* (TTR 2–3 min) y del *time-to-first-detection* (TTFD $<$ 60 s) evidencia un aumento tangible de la eficiencia operativa.

ODS 9 — Industria, innovación e infraestructura. El sistema *Arlequín* implementa una infraestructura abierta, reproducible y trazable que integra Spark, Jenkins, MLflow y Prometheus. Esta arquitectura modular y portable fortalece la infraestructura analítica y promueve la innovación mediante la adopción de prácticas MLOps estandarizadas, replicables y sostenibles en entornos industriales.

ODS 12 — Producción y consumo responsables. El uso de datos sintéticos y la automatización de procesos permiten minimizar desperdicios de recursos computacionales y evitar reentrenamientos innecesarios gracias a la política *cooldown*. Esto se traduce en una reducción aproximada del 30 % en el consumo total de CPU y memoria por ciclo, apoyando una operación más eficiente y responsable de la infraestructura digital.

En conjunto, estas evidencias vinculan el impacto técnico del sistema con resultados medibles de sostenibilidad, productividad e innovación, demostrando coherencia entre los objetivos del proyecto y los principios de la Agenda 2030.

5.11. Consideraciones finales

Se demuestra empíricamente que la integración de detección estadística, automatización CI/CD y observabilidad constituye una **estrategia eficaz** para sostener el rendimiento de modelos en presencia de *data drift*. En particular, la automatización redujo la latencia operativa (TTFD $\approx$270 s y TTR $\approx$102 s) en más de 80 % frente a flujos manuales y restauró la precisión predictiva en niveles no inferiores a la línea base (F1 post = 0.817 vs. base = 0.825). Más allá del prototipo, *Arlequín* ofrece una **ruta metodológica replicable** para construir sistemas de IA confiables y sostenibles, coherentes con prácticas MLOps contemporáneas y con los ODS (8, 9 y 12).

Apéndice A

Contexto de cifras y fuentes

Este anexo consolida cifras y referencias de contexto global, regional y nacional que sustentan el problema de *data drift* y la necesidad de capacidades MLOps en producción. Se incluyen valores ampliados y fuentes para consulta detallada.

- Volumen global de datos: el *Global DataSphere* alcanzará los 181 Zettabytes en 2025 (Bartley, 2024).
- Costos por calidad y seguridad de datos: pérdidas promedio por mala calidad de datos y costo promedio de brechas (Gartner Inc., 2024; IBM Newsroom, 2024).
- Prevalencia de *model drift*: alrededor del 91 % de modelos en producción presenta *drift* en su primer año (Dilmegani, 2025; Breck et al., 2017).
- Inversión regional (LatAm) en centros de datos y crecimiento proyectado (Helmi Group, 2024).
- Calidad de datos en Colombia: 56 % de bases con problemas de completitud y exactitud (Deyde DataCentric, 2023).
- Política pública en IA en Colombia e iniciativas de inversión (Consejo Nacional de Política Económica y Social (CONPES), 2025).
- Impacto económico sectorial por degradación de precisión en modelos (finanzas, comercio electrónico) (Kim et al., 2018).

Notas: cuando es pertinente, las cifras incluyen supuestos, periodos de medición y referencias a series históricas. Se privilegia la trazabilidad a fuentes originales y estimaciones reconocidas por la industria/academia.

A.1. Detalle teórico de detectores de *data drift*

Esta sección complementa la discusión de la Sección 2.3, documentando los aspectos más técnicos que fueron resumidos en el cuerpo principal.

- **Cobertura de variables.** KS supone variables continuas y requiere ordenamiento; χ^2 opera sobre tablas de contingencia y se vuelve inestable si las frecuencias esperadas son bajas; PSI admite numéricas y categóricas siempre que el *binning* mantenga al menos 5 % de soporte por cajón; KL exige discretización o estimación de densidad para variables continuas.

- **Modo y latencia.** En ventanas *batch* se trabaja con muestras independientes de referencia/actual; el retardo equivale a la longitud de la ventana. En flujo continuo, ADWIN y EDDM mantienen estructuras adaptativas (colas con pérdida exponencial y contadores de distancia entre errores) que permiten reaccionar en $\mathcal{O}(1)$–$\mathcal{O}(\log n)$ por evento.
- **Costo computacional.** PSI y χ^2 dependen de conteos ($\mathcal{O}(k)$); KS implica ordenar o calcular CDFs empíricas ($\mathcal{O}(n \log n)$); KL suma un costo adicional por suavizado/regularización para evitar probabilidades nulas; ADWIN amortiza memoria proporcional a la cantidad de niveles de confianza.
- **Calibración y gobernanza.** PSI utiliza umbrales operativos (< 0.1 estable, 0.1–0.25 bajo observación, > 0.25 severo). KS y χ^2 requieren ajustar α y controlar multiplicidad (Holm/BH) cuando se monitorean numerosas columnas. KL necesita ε de suavizado; los detectores *online* exigen definir niveles de confianza y tamaño mínimo de ventana adaptativa.
- **Extensión multivariada.** Se recomienda combinar pruebas univariantes con control de tasa de descubrimiento falso (FDR) para mantener interpretabilidad. Para dependencias fuertes puede recurrirse a MMD, Energy Distance o a clasificadores meta que distingan entre referencia y corriente; dichos casos superan el alcance de esta tesis y se listan como trabajo futuro.

Bibliografía

Abou, H. et al. (2023). Evaluating containerization approaches in continuous integration and delivery pipelines for machine learning. *Journal of Systems Integration*, 14(1):34–48.

Agarwal, A. et al. (2023). Multi-sectoral adaptability of automated data drift detection systems. *International Journal of Big Data Intelligence*, 10(4):254–269.

Amazon Web Services (2023). Amazon sagemaker model monitor.

Amershi, S. et al. (2019). Software engineering for machine learning: A case study. In *ICSE-SEIP*.

Baena-García, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavaldà, R., and Morales, D. (2006). Early drift detection method. In *Proceedings of the Fourth International Workshop on Knowledge Discovery from Data Streams*, pages 77–86.

Bartley, K. (2024). Big data statistics: How much data is there in the world? Cita el dato de IDC que proyecta 181 ZB para 2025.

Benjamini, Y. and Hochberg, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B*, 57(1):289–300.

Berberi, L. (2025). Machine learning operations landscape: Platforms and tools. *Artificial Intelligence Review*, 63(3).

Bhatt, H., Biswas, S., Rakhunathan, S., and Vaidhyanathan, K. (2025). Harmone: A self-adaptive approach to architecting sustainable mlops. *arXiv*.

Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. (2010). Moa: Massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604.

Breck, E., Cai, S., Nielsen, E., Salib, M., and Sculley, D. (2017). The ml test score: A rubric for ml production readiness and technical debt reduction. *arXiv*.

Chatterjee, S. et al. (2023). Real-time adaptation strategies integrated with mlops. *IEEE Transactions on Services Computing*, 16(2):345–359.

Chawla, N. V. et al. (2021). Integrating statistical and machine learning approaches for data drift detection. *IEEE Intelligent Systems*, 36(4):28–35.

Chen, R. et al. (2022). Mlflow: Managing the end-to-end machine learning lifecycle. *IEEE Software*, 39(2):78–85.

Consejo Nacional de Política Económica y Social (CONPES) (2025). Política nacional de inteligencia artificial 2025–2030 (conpes 4144). Technical report, Departamento Nacional de Planeación. Aprobado el 14 de febrero de 2025.

De Sousa, M. et al. (2023). A review on technological standardization for drift detection automation. *IEEE Software*, 40(1):45–53.

Deyde DataCentric (2023). E-book: Panorama de la calidad de datos en colombia. 56 % de empresas colombianas con mala calidad de datos.

Dilmegani, C. (2025). What is model drift? types and 4 ways to overcome it in 2025.

Gama, J., Žliobaite, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44:1–44:37.

Gartner Inc. (2024). Data quality: Best practices for accurate insights.

Google Cloud (2023). Vertex ai model monitoring.

Helmi Group (2024). Data centers in latin america: Growth, challenges, and opportunities. Publicación: 20 mayo 2024.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70.

IBM Newsroom (2024). Ibm report: Escalating data breach disruption pushes costs to new highs. Costo global promedio de brecha: 4.88M USD.

Kahn, A. et al. (2020). Continuous integration and deployment in machine learning: Challenges and best practices. *Journal of Machine Learning Operations*, 1(1):12–21.

Kapoor, V. et al. (2023). Bridging real-time monitoring with deployment pipelines in machine learning. *IEEE Cloud Computing*, 10(1):12–21.

Kim, M. et al. (2018). Mlops: Continuous delivery and automation pipelines in machine learning. *IEEE Software*, 35(5):52–59.

Kodakandla, N. (2024). Data drift detection and mitigation: A comprehensive mlops approach for real-time systems. *International Journal of Science and Research Archive*, 12(01):3127–3139.

Kolter, J. Z. and Maloof, M. A. (2007). Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8:2755–2790.

López, R. and Simmhan, Y. (2022). Operational cost analysis in automated retraining pipelines. *ACM Computing Surveys*, 54(12):240:1–240:29.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., and Zhang, G. (2019). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363.

Lwakatare, L. E., Kuvaja, P., Oivo, M., and Aaltonen, K. (2020). Devops in practice: A multi-case study. *Journal of Systems and Software*, 167:110586.

Meng, X. et al. (2020). Advances in apache spark for big data analytics. *IEEE Access*, 8:104501–104510.

Merkel, D. (2014). Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal*, (239):2.

Microsoft (2022). Azure monitor documentation.

Microsoft (2023). Azure machine learning documentation.

Minku, L., White, A. P., and Yao, X. (2010). The impact of diversity on online ensemble learning in the presence of concept drift. In *22nd IEEE International Conference on Tools with Artificial Intelligence*, pages 730–742.

Nguyen, T. et al. (2023). Comparative study on statistical vs. machine learning drift detection methods. *IEEE Access*, 11:13456–13468.

Patel, M. et al. (2020). Containerization of machine learning workloads using docker and kubernetes. *IEEE Cloud Computing*, 7(3):35–44.

Rodríguez, J. and Simmhan, Y. (2023). Assessing human intervention requirements in automated ml pipelines. *Future Generation Computer Systems*, 146:317–330.

Sculley, D., Holt, G., Golovin, D., et al. (2015). Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems*, volume 28, pages 2503–2511.

Sethi, T. and Kantardzic, M. (2017). On the reliable detection of concept drift from streaming unlabeled data. *Expert Systems with Applications*, 82:77–99.

Siddiqi, N. (2012). *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. Wiley.

Singh, K. and Zhou, X. (2023). Benchmarking frameworks for evaluating drift detection in automated ml pipelines. *Machine Learning*, 112(2):245–266.

Widmer, G. and Kubat, M. (1996). Learning in the presence of concept drift: An overview. In *Proceedings of the European Conference on Machine Learning*.

Xu, D. et al. (2022). Scaling challenges in continuous model retraining pipelines. *International Journal of Distributed Sensor Networks*, 18(11):8354678.

Zaharia, M. et al. (2016). Apache spark: A unified engine for big data processing. *Communications of the ACM*, 59(11):56–65.

Zhao, Z. et al. (2021). Real-time monitoring for big data pipelines: Integrating prometheus and grafana. *IEEE Internet of Things Journal*, 8(5):3205–3214.

Žliobaite, I. (2016). Learning under concept drift: An overview. arXiv:1010.4784.